Uma Arquitetura para Acesso e Integração de Dados em Sistemas Sensíveis ao Contexto

Natália Quirino de Oliveira

Vitória, ES
Outubro de 2007

Uma Arquitetura para Acesso e Integração de Dados em Sistemas Sensíveis ao Contexto

Natália Quirino de Oliveira

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática, na área de concentração em Banco de Dados.

Orientador:
Prof. Dr. Alvaro C. P. Barbosa

UNIVERSIDADE FEDERAL DO ESPIRITO SANTO – UFES
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Vitória – ES

2007

Uma Arquitetura para Acesso e Integração de Dados em Sistemas Sensíveis ao Contexto

COMISSÃO EXAMINADORA

_____

Prof. Dr. Alvaro Cesar Pereira Barbosa
Universidade Federal do Espírito Santo
Orientador

_____

Prof. Dr. Clever Ricardo Guareis de Farias
Universidade de São Paulo
Membro Externo

_____

Prof. Dr. José Gonçalves Pereira Filho
Universidade Federal do Espírito Santo
Membro Interno

Vitória, ES
Outubro de 2007

Dados Internacionais de Catalogação-na-publicação (CIP)
(Biblioteca Central da Universidade Federal do Espírito Santo, ES, Brasil)

Oliveira, Natália Quirino de, 1983-
O48a Uma arquitetura para acesso e integração de dados em sistemas sensíveis ao contexto / Natália Quirino de Oliveira. – 2007.
160 f. : il.

Orientador: Álvaro Cesar Pereira Barbosa.
Dissertação (mestrado) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Banco de dados distribuído. 2. Sistemas operacionais distribuídos (Computadores). 3. Serviços na Web. 4. Linguagem de consulta (Banco de dados). I. Barbosa, Álvaro Cesar Pereira. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título.

CDU: 004

Dedicatória

À minha família, por todo o carinho e auxílio recebido durante toda a jornada

Agradecimentos

Agradeço ao meu orientador pelos ensinamentos acadêmicos, ideais incentivados e amor à pesquisa. Sem ele, não teria um objetivo definido nem foco para alcançá-lo.

Aos demais professores do Departamento de Informática da UFES e colegas de curso por todo o auxílio e pela agradável convivência diária, em especial no LPRM.

Minha família, pelo incentivo, auxílio e compreensão, além ensinar os valores que norteiam minha vida.

Meus amigos, pelos bons momentos e pela paciência durante a realização do trabalho.

Ao CNPq pelo apoio financeiro.

Resumo

Sistemas de integração de dados vêm sendo desenvolvidos na tentativa de fornecer aos usuários informações consolidadas com transparência de distribuição e formato de armazenamento, e sua presença vem se tornando crucial. Em um cenário de Computação Ubíqua, as aplicações manipulam novos tipos de dados, de natureza contextual e dinâmica, o que dificulta o processo de integração. Este trabalho propõe uma arquitetura conceitual para acesso e integração de dados em ambientes móveis e sensíveis ao contexto. Um estudo de caso em tele-medicina ilustra as funcionalidades da arquitetura proposta.

Palavras-chave: Sistemas Distribuídos, Bancos de dados distribuídos, Integração de Dados, Computação Móvel, Computação Ubíqua e Sensível ao Contexto, Processamento de Consultas, Máquina de Execução de Consultas, Serviços Web.

Abstract

Data integration systems have been used for providing to the users information regardless of distribution and storage format, and their presence have became crucial on Information Technology. On a Ubiquitous Computing scenario, applications manipulate new types of data, with contextual and dynamic behavior, what makes integration process more difficult and complex. This work proposes a conceptual architecture for data access and integration for mobile and context-aware systems. A scenario on Tele medicine shows the advantages and functionalities of proposed architecture.

Keywords: Distributed Systems, Distributed Databases, Data Integration, Mobile Computing, Ubiquitous and Context-Aware Computing, Query Processing, Query Execution Engine, Web Services.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API	*Application Programmer Interface*
CoDIMS	*Configurable Data Integration Middleware System*
CORBA	*Common Object Request Broker Architecture*
DAML	*Darpa Agent markup Language*
DI	Departamento de Informática
DOM	*Document Object Model*
DTD	Definição de Tipo de Documento
FTP	*File Transfer Protocol*
GPS	*Global Positioning System*
HTML	*Hypertext Markup Language*
IA	Inteligência Artificial
J2ME	*Java 2 Micro Edition*
JDOM	*Java Domain Object Model*
JDK	*Java Development Ki*
JVM	*Java Virtual Machine t*
LPRM	*Laboratório de Pesquisas em Rede e Multimídia*
MEC	*Máquina de Execução de Consulta*
MIDP	*Mobile Information Device Profile*
OWL	*Web Ontology language*
QoS	*Quality of Service*
RAM	*Random Acess Memory*
RDF	*Resource Description Framework*
RDQL	*Resource Description Query Language*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Call*
SAX	*Simple API for XML*
SGBD	Sistema gerenciador de banco de dados
SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
UDDI	*Universal Description, Discovery and Integration*

UFES	*Universidade Federal do Espírito Santo*
URI	*Universal Resource Identifier*
UML	*Unified Modeling Language*
XML	*Extensible Markup Language*
W3C	*World Wide Web Consortium*
WSDL	*Web Service Definition Language*
WWW	*World Wide Web*

Sumário

1 Introdução

1.1 Contextualização

A última década trouxe grandes avanços nas tecnologias de computação e de comunicação móvel. Existe atualmente uma tendência de mudança no paradigma de computação, tradicionalmente estático, relativamente previsível e baseado em estações de trabalho, para um novo paradigma, altamente dinâmico, com constantes mudanças de ambiente causadas pela mobilidade de dispositivos tais como celulares, PDAs, GPS e sensores em geral.

Essa mudança é mais um passo em direção ao conceito de Computação Ubíqua (*Ubiquitous/Pervasive Computing*) introduzido por Mark Weiser [WEISER, 1991]. Weiser vislumbrou novos sistemas e ambientes acrescidos de recursos computacionais capazes de prover serviços e informações quando e onde sejam desejados pelos usuários ("*everywhere, everytime computing*"). Weiser propõe, assim, uma integração contínua entre ambiente e tecnologia na tarefa de auxiliar os usuários nas suas mais variadas atividades cotidianas.

A computação ubíqua sugere o desenvolvimento de novas aplicações que explorem as mudanças de contexto dentro de um domínio dinâmico. Essas aplicações seriam programadas para utilizar informações contextuais para dinamicamente selecionar ou executar as ações que melhor atendam às necessidades dos seus usuários. Diferentemente das aplicações tradicionais, essa nova categoria de aplicações, denominadas de "aplicações sensíveis ao contexto" (*context-aware applications*), levam em consideração na sua tomada de decisão e em seus processamentos, mais do que apenas as entradas de dados fornecidas explicitamente pelos usuários. Estas aplicações utilizam também entradas implícitas referentes ao contexto físico e computacional dos usuários e dos ambientes que os cercam. Assim, ao invés de tratar a mobilidade como um problema, as aplicações "*context-aware*" exploram a natureza contextual provocada pela mobilidade do usuário, com a clara intenção de produzir serviços mais flexíveis, adaptáveis, ricos em funcionalidade e centrados no usuário [KAPSAMMER *et. al.*, 2005][SINDEREN, 2006].

Aplicações sensíveis ao contexto vêm sendo desenvolvidas para vários domínios, tais como telemedicina, turismo, ambiental, segurança pública, universidades, rastreamento de cargas e veículos.

De forma a possibilitar essa característica dinâmica, torna-se necessária a disponibilidade e uso de dados contextuais, oriundos de diversos dispositivos tais como GPS e sensores em geral, além dos bancos de dados tradicionais. Assim, o desenvolvimento de aplicações sensíveis ao contexto demanda a existência de uma infra-estrutura de integração de dados e de estratégias de processamento de consultas sobre dados provenientes de ambientes dinâmicos, distribuídos e heterogêneos, para distintas tecnologias de comunicação e transmissão, configurações e capacidades.

Por sua vez, sistemas de integração de dados vêm sendo desenvolvidos, na tentativa de prover aos usuários uma visão única, uniforme e homogênea, para um grande número de fontes de dados heterogêneas, desenvolvidas independentemente. O objetivo maior desses sistemas é liberar o usuário de ter que localizar as fontes de dados, interagir com cada uma isoladamente e combinar os dados das múltiplas fontes de dados manualmente [HAKIMPOUR AND GEPPERT, 2001; HALEVY, 2003].

Desde o surgimento dos sistemas de gerenciamento de banco de dados, o problema de integração de dados tem sido reconhecido como ubíquo e criticamente importante [MILLER *et. al.*, 2001], e vem demandando pesquisa na área de Banco de Dados por mais de uma década [SHETH, 1990; HALEVY, 2003]. Este tema ainda é atual, sendo renovado por novas conjunturas, como o amplo uso da *Web*, sendo que [ZIEGLER; DITTRICH, 2004] e [ABITEBOUL *et. al.*, 2005] enfatizam o potencial da integração de dados utilizada em conjunto com tecnologias emergentes.

Considerando o exposto acima, pode-se perceber que se torna natural a incorporação da integração de dados no escopo do desenvolvimento das aplicações sensíveis ao contexto, permitindo uma visão única e combinação entre vários tipos de contexto que se encontram distribuídos.

A Sociedade Brasileira da Computação (SBC) em seu documento “os Grandes Desafios da Computação no Brasil, a partir do modelo dos eventos internacionais existentes” [SBC, 2006], reconhece a relevância da computação sensível ao contexto e a integração de dados durante os próximos anos. Um dos desafios consiste na Gestão da Informação em grandes volumes de dados multimídia distribuídos. A justificativa para tal é a geração exponencial de dados causada pelo surgimento de dispositivos que capturam novos tipos de dados extremamente complexos – satélites, micros-sensores, telescópios, câmeras de vídeo.

Finalmente, dados são gerados por cientistas e pesquisadores ao fazer experimentos. Alguns dos grandes problemas técnicos e científicos apontados para fazer frente a este desafio são:

- Definição e uso da noção de contexto para a recuperação de informação, considerando fatores como localização do usuário, perfil de interesses, objetivos dentre outros;
- Consideração, no armazenamento e recuperação, de fatores inerentes à heterogeneidade na aquisição de dados tais como fatores temporais e culturais, mas também tecnológicos, como sensores, celulares, PDAs.

1.2 Motivação

Este trabalho teve seu início durante o desenvolvimento do projeto DBMWare: Infraestrutura de Suporte ao Desenvolvimento Baseado em Modelos de Aplicações *Context-Aware* [DBMWARE, 2005]. O objetivo do DBMWare foi propor uma metodologia para o desenvolvimento de aplicações distribuídas segundo o enfoque MDA (*Model Driven Architecture*), oferecendo assim um ambiente adequado à construção e execução desse tipo de aplicações, nos mais diferentes domínios. Especificamente, o enfoque do DBMWare visou definir uma nova arquitetura para uma plataforma de suporte à execução de aplicações sensíveis ao contexto.

A partir do conhecimento adquirido com o desenvolvimento do DBMWare foi iniciado um novo projeto denominado Infraware: Uma Plataforma para Desenvolvimento de Aplicações Móveis e *Context-Aware* [PEREIRA FILHO *et. al.*, 2005] [PEREIRA FILHO *et. al.*, 2006] [PEREIRA FILHO *et. al.*, 2006b]. Seu objetivo foi desenvolver uma plataforma de serviços sensíveis ao contexto, baseada em *web services*, para suportar a execução de aplicações móveis sensíveis ao contexto.

Nestes dois projetos uma necessidade comum foi pesquisar e desenvolver funcionalidades para permitir a integração de dados oriundos de diversos dispositivos chamados de provedores de contexto. Os provedores de contexto se caracterizam pelo dinamismo, mobilidade, validade temporal, validade geográfica, dentre outros fatores, que os diferenciam das fontes de dados tradicionais. Além disso, devido ao aspecto pró-ativo das aplicações sensíveis ao contexto, os dados devem ser entregues quando eventos subscritos pelo usuário são identificados.

A integração de dados contextuais heterogêneos e distribuídos oriundos de sensores e outras fontes é a grande motivação deste trabalho. Em tal cenário, integração de dados, embora já reconhecida como criticamente importante na área de banco de dados e já devidamente pesquisada em seu aspecto tradicional, pode ser ampliada para atender a esta nova área de aplicações sensíveis ao contexto, assumindo uma nova conotação e desafios distintos nesta nova conjuntura. Para tal, avanços em tecnologias de redes móveis e de sensores sem fio (RSSF) devem ser unidos aos esforços de modelagem de informações dinâmicas, para o surgimento de mecanismos e metodologias de acesso, aquisição e integração de dados contextuais.

1.3 Objetivo

Este trabalho propõe uma arquitetura para integração de dados em ambientes sensíveis ao contexto e uma estratégia para execução de consultas em tais ambientes por meio da especificação e implementação de um componente denominado Acesso e Integração de Dados. Da experiência adquirida anteriormente em pesquisas de integração de dados, especificamente no desenvolvimento do projeto CoDIMS [BARBOSA *et. al.*, 2002], para o desenvolvimento do componente Acesso e Integração de Dados foi gerada uma nova instância do *framework* CoDIMS, adaptado a este novo cenário. Devido ao relacionamento entre os componentes do CoDIMS, estes foram estendidos e configurados, originando uma nova arquitetura, apropriada para lidar com diferentes modos de requisições e formatos dos dados, requeridos por aplicações sensíveis ao contexto.

1.4 Contribuições esperadas

Os desafios apontados vão ao encontro aos estudos sobre integração de dados realizados neste trabalho, no âmbito dos projetos supracitados. Apesar deste recente reconhecimento da importância desta área de pesquisa, como demonstrado no documento “Grandes desafios” da SBC, pouco enfoque tem sido dado na literatura ao acesso e integração de dados oriundos de fontes móveis para sistemas de computação sensível ao contexto e suas particularidades [BIERMAN *et. al.*, 2003] [PITOURA; STEFANDIS, 2004]. O CoDIMS se apresenta como uma alternativa viável para esta tarefa, devido à sua arquitetura. Entretanto, diversas modificações devem ser realizadas em seus componentes para tal propósito.

Os novos modos de requisições demandam entrega *push* de dados para as plataformas DBMWARE/Infraware, através de uma sintaxe bem definida; dados estes a serem reconhecidos de diversos tipos de fontes contextuais através de metadados adaptados ao domínio da computação sensível ao contexto; finalmente, estes devem ser armazenados para uso posterior.

Segundo [ABITEBOUL *et. al.*, 2005] é importante explorar novas oportunidades para utilizar tecnologias de banco de dados em novas aplicações. Especificamente, estas podem ser ampliadas para atender às aplicações sensíveis ao contexto. Assim, decidimos estender nossa experiência em sistemas de integração de dados e investigar novas abordagens a serem utilizadas em ambientes sensíveis ao contexto.

As seguintes contribuições são esperadas para o módulo de acesso e integração de dados das plataformas DBMWare/Infraware, e também à literatura disponível:

- Uma lista de novos requisitos a serem atendidos por sistemas de acesso, integração e manipulação de dados para aplicações sensíveis ao contexto. A partir desta lista foi possível definir um ambiente que possa atender às necessidades da computação sensível a contexto;
- Apresentação de uma arquitetura que facilite o reuso e a adaptação de seus componentes para utilização em diferentes domínios de aplicação, que permita a entrega de dados em diferentes formatos, de acordo com suas necessidades. Além disso, o baixo acoplamento permite que novos requisitos possam ser incorporados;
- Especificação de um módulo adaptável para a detecção distribuída de eventos através de uma máquina de execução de consultas distribuída. A sintaxe e algoritmos da MEC podem ser substituídos contanto que suas interfaces sejam respeitadas;
- Especificação de um módulo para o armazenamento de eventos ocorridos (*event report*), que podem ser utilizados futuramente para fins de consulta e descoberta de padrões;
- Utilização de metadados, que facilitam a descrição e adição de novos componentes, fontes, consultas, eventos, perfis de usuário, estatísticas de acesso e tradução dos resultados de consultas para diversos formatos;
- Armazenamento configurável de histórico contextual próximo às fontes de dados para uso corrente, de acordo com as limitações encontradas por cada aplicação, além de um armazenamento secundário centralizado.

Devido ao fato da plataforma Infraware ser uma plataforma geral para aplicações sensíveis ao contexto em vários domínios, foi necessária a especificação de uma aplicação piloto. Esta aplicação será no domínio da tele medicina, e deu origem ao projeto TeleCardio [ANDREAO *et. al.*, 2006], do Departamento de Informática em conjunto com o Departamento de Engenharia Elétrica da UFES. Este domínio, portanto, será utilizado neste trabalho como cenário de uso.

1.5 Metodologia

Este trabalho iniciou-se com o levantamento e leitura da bibliografia relacionada, com o intuito de estudar a área de arquiteturas de sistemas sensíveis ao contexto e a de integração de dados heterogêneos.

Vários estudos e discussões foram feitos no grupo de pesquisa do Laboratório de Pesquisa em Redes e Multimídia (LPRM), do Departamento de Informática da UFES, tendo em vista o desenvolvimento dos projetos DBMWARE, INFRAWARE, TELECARDIO e CoDIMS, com seminários, relatórios técnicos e publicações aceitas. A partir desses estudos e discussões, amadureceu a idéia que culminou com a presente dissertação.

A metodologia empregada no desenvolvimento do projeto inclui:

- Definição (especificação, projeto e implementação) da arquitetura proposta. Foram estudados os diagramas UML (*Unified Modeling Language*) [UML, 2006] para se gerar os modelos do módulo proposto, além de bibliotecas para tornar possível a implementação do ambiente;
- Elaboração de um estudo de caso, que se baseia na proposta de [MENEQUINI, 2006] para validar a incorporação do Módulo de Acesso e Integração de Dados aos projetos DBMWARE / INFRAWARE / TELECARDIO;
- Estudo de tecnologias relacionadas, tais como *web services* e OWL;
- Redação de um conjunto de Relatórios Técnicos em temas relacionados ao projeto;
- Leitura e análise de trabalhos relacionados.

1.6 Organização do Trabalho

O restante deste trabalho está organizado da seguinte forma: o Capítulo 2 apresenta conceitos e tecnologias necessários ao entendimento do problema, como integração de dados e

computação sensível ao contexto. O Capítulo 3 apresenta os requisitos para a integração de dados contextuais. O Capítulo 4 apresenta trabalhos relacionados, listados e comparados de acordo com os requisitos apontados anteriormente. O Capítulo 5 apresenta a modelagem conceitual e o projeto da arquitetura, baseado em diagramas UML. O Capítulo 6 apresenta o estudo de caso a aplicação prova de conceito no domínio hospitalar, além da implementação e interfaces com o usuário. Finalmente, o Capítulo 7 apresenta conclusões, contribuições e trabalhos futuros.

2 Conceitos e Tecnologias

Neste capítulo são apresentados os conceitos básicos para o desenvolvimento de um *middleware* para integração de dados em ambientes sensíveis ao contexto. Estão incluídos os conceitos de sensibilidade ao contexto, desenvolvimento de *software*, em particular *frameworks* e componentes, bem como conceitos de integração de dados e de banco de dados. São também apresentadas algumas tecnologias e linguagens utilizadas na fase de implementação descrita no Capítulo 6.

2.1 Sensibilidade ao Contexto

Existem várias definições de contexto na literatura. A noção de contexto foi introduzida primeiramente por [SCHILIT *et al.*, 1994], que o descrevia como a localização, proximidade de pessoas e objetos, assim como mudanças com o tempo. [SCHILIT *et al*., 1994] declarou que há três importantes aspectos de contexto: "onde uma pessoa se encontra, com quem ela está e quais recursos estão próximos".

[DEY, 2001] sustentou que essas definições são muito específicas, propondo então a definição mais utilizada para contexto:

> *"Contexto é qualquer informação que pode ser usada para caracterizar uma situação de uma entidade. Uma entidade é uma pessoa, um lugar ou um objeto que é considerado relevante para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a própria aplicação."*

Ainda segundo [DEY, 2001], existe mais uma categoria essencial de contexto: o tempo, que além de fornecer o instante ou período do fato, possibilita também o levantamento de importantes informações históricas.

Esta definição, entretanto, é por vezes considerada demasiadamente generalista. Algumas classificações tentam limitar os tipos possíveis de contexto, englobando fatores ambientais, sociais, atividades realizadas pelos atores e agenda de compromissos. Neste sentido, todas as informações dinâmicas de um ambiente de trabalho, como a temperatura de uma sala, reuniões marcadas por um funcionário e seu *status* atual (ocupado/horário de almoço) são consideradas o contexto daquele exato momento.

Várias abordagens têm sido apresentadas na literatura para a representação de contexto. Uma delas é a linguagem UML-CW, encontrada no projeto DBMWare

[DBMWARE, 2005], representada na Figura 1. Esta figura mostra as associações (*CWAssociation*) possíveis entre entidades (*CWEntity*), ou entre entidades e atributos, que podem ser classificados como estático, dinâmicos, de sensores, perfil e demais dados derivados.

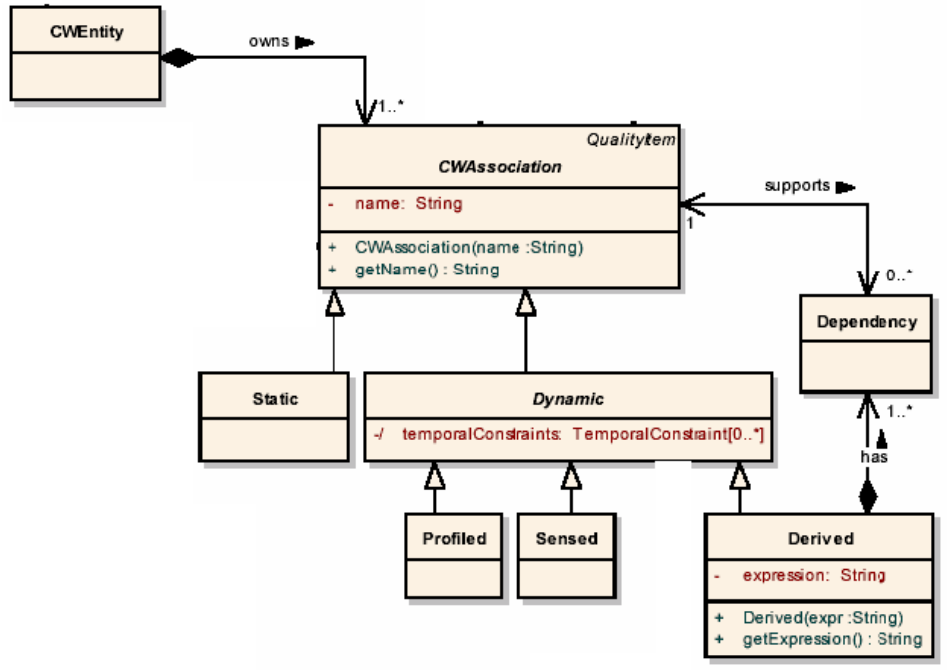

Figura 1: Metamodelo de contexto UML-CW (extraído de [DBMAWRE, 2005]).

Assim como existem várias definições de contexto, há, também, muitas tentativas de se definir sensibilidade ao contexto (*context-aware*) e de descrever como o contexto pode ser usado pelas aplicações. Uma definição mais geral de computação sensível ao contexto, dada por [DEY, 2001]:

> *"Um sistema é sensível ao contexto se ele usa contexto para fornecer informações relevantes e/ou serviços para o usuário, onde a relevância depende da tarefa do usuário.*

Segundo [PASCOE, 1998], sensibilidade ao contexto é a habilidade de dispositivos computacionais de detectar, sentir, interpretar e responder aos aspectos relacionados ao ambiente do usuário e aos próprios dispositivos. Muitos definem aplicações sensíveis ao contexto como sendo aplicações que mudam dinamicamente ou adaptam seu comportamento de acordo com o contexto da aplicação sem a necessidade de entradas explícitas pelo usuário. Um exemplo típico é o de um viajante que recebe automaticamente informações sobre o local onde se encontra. É enriquecida, assim, a tomada de decisões de uma aplicação e sua

interação com o usuário final. A Figura 2 mostra a distinção entre sistemas convencionais e sistemas sensíveis ao contexto, com entradas implícitas, sem a intervenção humana.

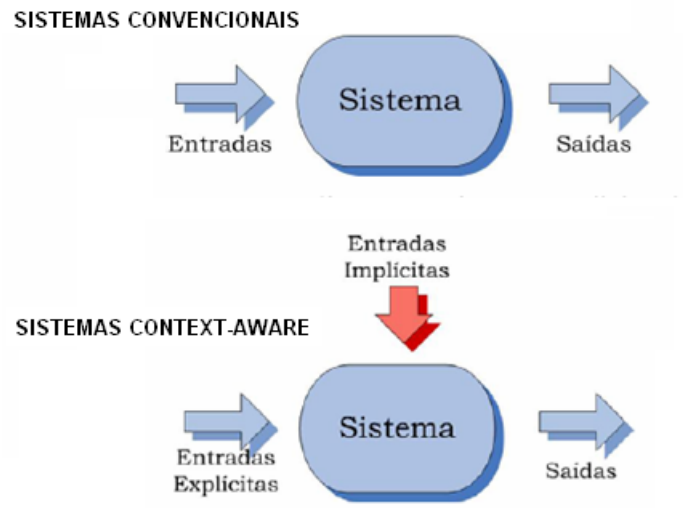

Figura 2: Sistemas convencionais X *sistemas context-aware*

As aplicações sensíveis ao contexto, também conhecidas como "computação pró-ativa" devem facilitar a automatização de tarefas e o disparo de ações quando o contexto de uma aplicação muda. De fato, pode-se pensar em uma aplicação sensível ao contexto como um conjunto de eventos relativos ao ambiente que devem desencadear processamento de interesse do usuário. Tal paradigma vislumbra novas possibilidades de aplicações, entretanto, apresenta também requisitos e desafios a serem solucionados [PEREIRA FILHO et *al.*, 2006].

Alguns dos cenários que se destacam em pesquisas sendo realizadas e outros que são potencialmente promissores, são listados a seguir. Estes cenários caracterizam-se por serem altamente dinâmicos, por apresentarem mudanças contextuais em curtos intervalos de tempo, levando as aplicações a tomarem decisões rápidas de acordo com as variações de contexto. Assim, cada tipo de aplicação toma suas decisões e realiza ações de acordo com o ambiente e com a vontade de seus usuários. Alguns exemplos de potenciais aplicações são:

• *Aplicações turísticas*. Mapas de atrações turísticas e posicionamento relativo dos turistas em relação às atrações que desejam visitar. As atrações que são visualizadas variam de acordo com o perfil de cada usuário;

• *Aplicações policiais*. Mapeamento e controle de viaturas e policiais em situações do dia-a-dia ou em situações mais perigosas como uma perseguição a um fugitivo;

• *Aplicações voltadas ao meio ambiente*. Controle e monitoramento de diversos animais ao mesmo tempo e em grandes áreas para fins de pesquisa e preservação do meio ambiente;

• *Ambiente acadêmico e mapas de universidades.* Mapas universitários e localização de alunos e professores em seu ambiente de trabalho;

• *Rastreamento de veículos em transporte de cargas e mercadorias.* Monitoramento do posicionamento de veículos durante o percurso de um frete;

Visto, portanto, o conceito e escopo das aplicações sensíveis ao contexto, a seguir são mostradas abordagens que podem ser ou têm sido utilizadas para o desenvolvimento de infra-estrutura para tais aplicações, considerando o dinamismo, necessidades de atualização e variedade de tais aplicações.

2.2 Desenvolvimento de Sistemas de Software

2.2.1 *Frameworks*

Um *framework* é uma arquitetura semi-completa que pode ser instanciada para produzir aplicações customizadas [FAYAD *et. al.*, 1999], ou seja, é uma arquitetura reutilizável. Seu uso resulta em flexibilidade e rapidez no atendimento de requisitos de novas aplicações. Embora a construção de sistemas a partir de outros já existentes seja prática comum, não é levado em consideração, durante o projeto, a possibilidade de sua reutilização no futuro, nem a possibilidade de integração com outros sistemas. Em um *framework*, torna-se necessária a diferenciação entre suas partes fixas e partes adaptáveis (*hotspots*).

Quando usados em conjunto com *patterns*, bibliotecas de classes e componentes, os *frameworks* podem, significativamente, aumentar a qualidade do *software* e reduzir o esforço de desenvolvimento. *Frameworks* são mais customizáveis que componentes e têm interfaces mais complexas. *Frameworks* e componentes são tecnologias diferentes, mas que podem ser complementares. Os *frameworks* podem auxiliar o desenvolvimento de infra-estrutura para aplicações sensíveis ao contexto, ao possibilitar a adaptação a vários domínios.

2.2.2 Desenvolvimento Baseado em Componentes

O desenvolvimento baseado em componentes tem como objetivo a reutilização de uma solução já testada e de eficácia comprovada, constituindo uma excelente maneira de se ter *software* testado evitando-se repetir etapas já realizadas. Estas soluções podem ser agrupadas em bibliotecas de componentes apropriados para reuso, contendo código, análise, modelagem

e testes. Eles somente se comunicam através de suas interfaces, e o usuário não precisa supor nada sobre sua representação interna.

Segundo [PRESSMAN, 2001], uma das atividades a serem consideradas cuidadosamente é o projeto das interfaces. Também é ressaltada a facilidade de uso e é feita uma analogia com produtos de engenharia, como eletro-eletrônicos, simplificada com os formatos padronizados.

O desenvolvimento por componentes pode auxiliar o desenvolvimento de aplicações sensíveis ao contexto ao possibilitar bibliotecas reutilizáveis para funcionalidades da computação sensível ao contexto.

2.2.3 Arquitetura em Camadas

O desenvolvimento de arquitetura em camadas busca o baixo acoplamento entre funcionalidades de *software* [PRESSMAN, 2001]. Uma das vantagens atingidas com este padrão é que cada camada se comunica somente com as camadas imediatamente inferior e superior. Assim, pode-se inferir que nenhum módulo de um sistema pode trabalhar com duas camadas. O número de camadas ideal deve ser determinado levando-se em conta o equilíbrio entre *overhead* de excessivo número de camadas e facilidade de uso. Arquiteturas em camadas têm sido utilizadas para sistemas sensíveis ao contexto [COSTA, 2003], pois desacoplam tecnologias utilizadas em diversas camadas e facilitam mudanças posteriores.

2.2.4 *Web Services*

A tecnologia de *Web Services* [W3C, 2007] [*Web Service*, 2007] é baseada em padrões abertos, como XML, e surgiu da necessidade de comunicação e cooperação entre aplicações de maneira rápida, simples e barata, especialmente na *Internet*. Em organizações com aplicações heterogêneas e arquiteturas distribuídas, a introdução de *web services* padroniza a comunicação e propicia a interoperabilidade de aplicativos escritos em diferentes linguagens de programação, disponibilizados através de diferentes plataformas de desenvolvimento, sistemas operacionais e tecnologias de rede. Esta comunicação é realizada utilizando-se mensagens XML. Cada *web service* é identificado unequivocamente por um URI.

As principais linguagens relativas aos *Web Services* são:

- WSDL (*Web Service Description Languagea*): descreve as funcionalidades oferecidas pelo *web service*;

- UDDI (*Universal Description Language*): linguagem para anúncio de *web services* em ambientes abertos para uso público;
- SOAP (*Simple Object Access Protocol*): linguagem para comunicação entre aplicação cliente e *web service*.

As abordagens vistas anteriormente são viáveis para o projeto de sistemas de software. A seguir, são apresentados alguns conceitos e tecnologias para o escopo particular de integração de dados e de bancos de dados para embasar a arquitetura proposta neste trabalho.

2.3 Integração de Dados e Bancos de Dados

Muitas aplicações atualmente existentes necessitam incorporar informações de diversas fontes de dados onde os dados armazenados podem diferir na representação ou na tecnologia utilizada. As dificuldades de adaptá-los para se comunicarem e compartilharem informações com outros sistemas mais novos e avançados tecnologicamente são grandes e complexas. Muitos sistemas ainda existentes foram construídos e bem customizados para atender às necessidades específicas das empresas e, apesar de poderem estar desatualizados tecnologicamente, a importância dessas aplicações para a empresa continua impossível de ser ignorada. Assim, torna-se importante o desenvolvimento de sistemas que permitam o acesso e a integração de dados existentes em diferentes fontes [BARBOSA, 2001] [SHETH; LARSON, 1990].

A integração de dados, embora já reconhecida como criticamente importante na área de banco de dados e já devidamente pesquisada em seu aspecto tradicional, pode ser ampliada para atender novas aplicações [ABITEBOUL, 2003]. Em particular, estas novas aplicações podem ser sensíveis ao contexto, assumindo, dessa forma, uma nova conotação e enfrentando novos desafios inerentes a essa classe de aplicações. Para tal, avanços em tecnologias de redes móveis e de sensores sem fio (RSSF) devem ser unidos a esforços de modelagem de informações dinâmicas para o surgimento de mecanismos e metodologias de acesso, aquisição e integração de dados contextuais.

2.3.1 *Wrappers*

Em [GAMMA *et. al.*, 1995], um dos padrões de projetos estruturais descritos é o padrão *Adapter*, ou *Wrapper*. Ele possibilita o ajuste de interfaces entre classes distintas, permitindo,

dessa maneira, a comunicação entre classes que não poderiam trabalhar juntas devido à incompatibilidade de suas interfaces.

Nos sistemas de integração de dados, *wrappers* realizam a tradução entre o modelo de dados comum, utilizado no sistema integrador, e o modelo de dados de cada fonte. Ou seja, para ser possível a integração de dados deve-se homogenizar suas heterogeneidades. No caso específico de modelo de dados, estes devem ser convertidos para um único modelo, chamado modelo de dados comum. Por exemplo, se o modelo de dados comum é relacional e a intenção é integrar com fontes de dados XML, deve-se construir *wrappers* que traduzam XML para relacional [NADAI, 2006].

2.3.2 Metadados

O uso tradicional dos metadados em banco de dados e em sistemas de integração de dados é na descrição dos dados e das fontes de dados para serem utilizados internamente na execução de consultas. Nas aplicações sensíveis ao contexto, os metadados devem adicionalmente dispor de novas informações sobre a autoria, origem e atualidade dos dados, além de características de perfis dos usuários e dos recursos computacionais das fontes, fornecendo uma visão homogênea sobre os diferentes tipos de provedores de contexto.

Conforme visto em [SILVESTRE, 2005], "*A heterogeneidade ocasiona problemas especialmente no que diz respeito à Integração de Esquemas. Integração de Esquemas consiste justamente no processo de construção de uma visão homogênea, ou seja, prover um esquema integrado sobre os metadados pertencentes a fontes distribuídas e heterogêneas, através da qual os usuários podem realizar consultas como em bancos de dados centralizados. Uma solução para minimizar os problemas de heterogeneidade é a adição de mais semântica aos metadados.*" Para tanto, recentemente, ontologias vêm sendo utilizadas para representação de metadados, devido a seu poder de expressão. Um resumo sobre ontologias é apresentado a seguir.

2.3.2.1 Ontologias

Originalmente sendo um campo de estudo da Filosofia (o "estudo do ser enquanto ser"), as ontologias passaram a ser utilizadas pela comunidade de Inteligência Artificial (IA), e depois por outras comunidades da Ciência da Computação.

Uma ontologia, segundo [GRUBER, 1993], é uma "especificação explícita e formal de uma conceitualização compartilhada". Trata-se de um conjunto de conceitos e definições sobre um determinado domínio e também as relações entre tais conceitos. Freqüentemente, uma ontologia é referida como um vocabulário compartilhado e esta é precisamente sua maior função: obter consenso sobre os significados de termos em uma determinada área a fim de se transmitir conhecimento.

O uso de ontologias vem ganhando mais importância, não ficando mais restrito à comunidade de IA, como no passado. Um dos campos onde as ontologias terão uma importância vital é a Web Semântica, que necessita lidar com informações de várias fontes. Elas também têm sido utilizadas com sucesso pela comunidade de bancos de dados distribuídos [MENA *et. al.*, 2000] [SILVESTRE, 2005] e para a computação sensível ao contexto [CHEN, 2004].

Algumas vantagens no uso de ontologias:

- **Conhecimento compartilhado:** compartilhando-se um conjunto de termos, facilita-se a comunicação entre pessoas, organizações ou entre agentes computacionais, possibilitando ao agente a extração e integração de informações de diferentes fontes. No caso de aplicações sensíveis ao contexto, as ontologias podem ser responsáveis pela modelagem e representação de contexto, independente de como estejam representadas em cada provedor de contexto [CHEN, 2004];
- **Reutilização de conhecimento sobre um domínio:** freqüentemente, na modelagem de um sistema, não separamos as informações sobre o domínio das informações especificas à aplicação. Os resultados disto são retrabalho, nas demais aplicações dentro do mesmo domínio, e possibilidade de inconsistência;
- **Explicitar informações sobre o domínio:** Algumas vezes, algumas informações sobre o domínio são embutidas dentro do código, tornando-o menos legível. As ontologias são uma maneira natural e simples de se explicar um domínio;
- **Processamento por máquina:** sendo formais e não-ambíguas, as ontologias são processáveis por máquinas;
- **Possibilidade de se especificar sinônimos, siglas, traduções para idiomas distintos:** tais recursos facilitam a busca de informações, e a descrição de dados contextuais e busca por serviços oferecidos, sendo superior à busca por palavras chaves;
- **Possibilidade de inferência:** sendo as ontologias processáveis por máquina, podem-se

obter novos fatos não mencionados originalmente.

Apesar das vantagens mostradas anteriormente, não se deve assumir que uma ontologia cobre todos os aspectos desejáveis da modelagem do conhecimento por si só. Com a visão de semântica de várias áreas como a computação sensível ao contexto*, Web Semântica* e *grid*, as ontologias passam a ter uma grande importância, sendo utilizadas para funcionamento interno das arquiteturas, no armazenamento automático de metadados, processo também conhecido como "anotação semântica" e representação externa dos diversos provedores de contexto, como sensores.

As principais categorias de ontologias identificadas por [DE ROURE *et. al.*, 2001] são:

- **Ontologias de domínio:** conceitualização de objetos importantes, suas propriedades e relacionamentos. Como exemplo, temos um conjunto de termos para estudos de clima;
- **Ontologias de tarefa:** conceitualizações de tarefas e processos, por exemplo, conjunto de estágios de um processo de análise química;
- **Ontologias de qualidade:** indica características como a taxa de erros de um processo de análise médica de imagens, ou a precisão de um instrumento em um laboratório;
- **Ontologias de valor:** dão uma idéia definida da importância de um conteúdo, sua raridade, ou o custo de se executar um determinado processo;
- **Ontologias de personalização:** importantes para se estabelecer um perfil e expectativas do usuário, de acordo com a familiaridade do usuário com o domínio.

Cada instituição pode decidir pelo uso de diferentes modos de armazenamento de dados, devido a um paradigma preferido (relacional, orientado a objeto, objeto-relacional) ou questões econômicas. Sistemas computacionais podem utilizar várias tecnologias para o armazenamento de informações: arquivos de texto, bases de dados relacionais, arquivos XML, objetos RMI. Além disso, somam-se as diferenças de fabricantes (Oracle, DB2, *etc*.).

Uma solução para este problema é a publicação de metadados, que permitem aos dados serem encontradas por uma determinada aplicação. As ontologias podem ser utilizadas para a representação de fontes de dados, tanto sua infra-estrutura quanto seu conteúdo, de modo a fornecer transparência ao usuário e ao próprio sistema integrador.

2.3.3 Bancos de Dados Ativos

Um Sistema Gerenciador de Banco de Dados (SGBD) é um *software* confiável e eficaz na manutenção e operação de grandes bancos de dados, fornecendo funcionalidades como otimização de consultas, tolerância à falhas e controle de concorrência.

Tradicionalmente, os SGBDs, e também os sistemas de integração de dados, são passivos no sentido que os comandos são executados somente quando requisitados por uma aplicação; ou seja, não oferecem suporte para o gerenciamento automático sobre o estado do banco de dados em reposta a atualizações ocorridas. Para se contornar estes obstáculos, foram propostos os bancos de dados ativos, que nas últimas décadas têm sido uma área amplamente abordada pela literatura [WIDOW, 1996] [SCHWIDERSKI, 1996] [PATON; DIAZ, 1999] [BERNAUER *et. al.*, 2004]. Existem várias abordagens acadêmicas, e os maiores SGBDs comerciais atuais possuem funcionalidades de bancos de dados ativos [POSTGRES, 2007] [ORACLE, 2007];

Os bancos de dados ativos são sistemas de banco de dados estendidos com um sistema de regras. Tais sistemas são capazes de reconhecer eventos de interesse gerados quando ao término de uma transação, selecionar e ativar as regras correspondentes e, quando a condição for verdadeira, executar as ações correspondentes. O modelo mais utilizado para esse comportamento reativo são as regras E-C-A (Evento - Condição - Ação), visto na Figura 3. Este modelo, segundo [DAYAL *et. al.*, 1994], possui a forma:

On *evento* **If** *condição* **Then** *ação*

As partes que compõem a regra são as seguintes:

- **Evento** – alteração no estado do banco de dados que ativa a regra (“gatilho”);
- **Condição** - expressão que é avaliada quando um evento é detectado.
- **Ação** - o procedimento que é executado quando a condição for verdadeira.

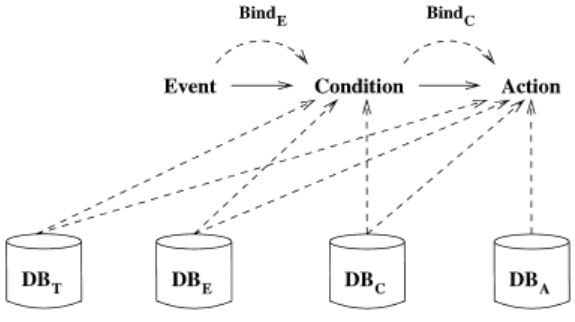

Figura 3: Componentes de uma regra ECA (extraído de [DAYAL *et al.*, 1994]).

No código a seguir, vê-se um exemplo em SQL das regras que especificam quando e quais ações deverão ser executadas: o cancelamento da operação de inserção de empregados caso seu salário seja superior ao de seu gerente.

```
CREATE TRIGGER empregado-salario
        ON INSERT empregado
        IF empregado.salario >
                    SELECT E.salario FROM empregado E
                    WHERE E.nome = empregado.gerente)
 THEN abort
```

Muitas arquiteturas possuem a diferenciação clara entre três fases: (*i*) detecção de evento, através de um detector; (*ii*) gerenciamento das condições; (*iii*) agendamento de ações, através de um módulo *scheduler*. A Figura 3 mostra que a detecção de eventos e o disparo de ações podem ser desvincualdos, permitindo uma ampla flexibilidade para, por exemplo, executar as ações em qualquer período após a ocorrência da condição, além de uma maior flexibilidade para implementação.

Extensões a estas regras podem ser criadas. O trabalho realizado em [HEIMRICH;SPECHT, 2002] apresenta o conceito de regras ECA extendidas. Nelas, não se assume que o valor de um evento é conhecido como verdadeiro ou falso, sendo criado um terceiro valor, Ω que representa "ocorrência indefinida". A partir disso, os eventos são representados por uma quádrupla (Evento, Condição, Ação e Ação alternativa). Segundo os autores, estas características auxiliam a detecção descentralizada de eventos. Em ambientes móveis e com frequentes problemas de conectividade devido a falhas em redes sem fio ou para economia de energia, esta é uma abordagem atrativa.

Tendo-se definido regras ECA, deve-se ter a definição de um evento. Segundo [CILIA, 2002], "um evento é uma ação instantânea e atômica", ou seja, ela ocorre um ponto especifico no tempo, não ocorrendo parcialmente. Formalmente, filtros devem ser restritos a predicados com resultado booleano. Tais eventos são denominados eventos simples. Um evento composto, entretanto, é formado por vários eventos simples, podendo possuir uma história de sua identificação. O primeiro evento simples identificado na ocorrência do evento composto é chamado "*initiator*". O último evento identificado, que engatilha a regra, é chamado "*terminator*" [CILIA, 2002]. A detecção de eventos deve sempre ter a mesma expressividade de um ambiente centralizado, sem ser afetados pela distribuição dos dados.

Existem vários tipos de eventos disparados em sistemas. Primeiramente, eventos baseados em tempo (*time-driven*). Existem eventos relacionados à modificação de dados

(criação ou deleção de tuplas), listagem de dados (cláusula *SELECT*) ou ao processamento de transações (ao se ocorrer o *commit* ou *rollback* de uma transação) [SCHWIDERSKI, 1996]. Na computação sensível ao contexto, um evento representa uma ação ocorrida no mundo real, com semântica e consequências. Estes eventos são chamados eventos abstratos [CILIA *et. al.*, 2003] e, por isso, são válidas linguagens em alto nível para a representação de eventos.

Um evento pode se encontrar em um dos seguintes estados [PIETZUCH *et. al.*, 2003]: (*i*) inicial, quando é criado; (*ii*) normal, ou seja, em espera; (*iii*) gerador, quando as condições são atendidas, e é aguardado o momento de execução da ação; (*iv*) disparo, quando a ação pode ser executada.

Em um ambiente distribuído, eventos devem ser decompostos em eventos simples, onde a condição de disparo de cada um destes apenas se refere a um dado (por exemplo, uma tabela relacional). O resultado final é uma árvore, onde os eventos simples são passados aos seus pais, que são eventos mais complexos. Esta arquitetura facilita e fornece melhor performance ao processamento paralelo de consultas [SCHWIDERSKI, 1996]. Ao mesmo tempo, o usuário realiza consultas em alto nível independentemente da distribuição das fontes. A estratégia de árvore também facilita a adaptação a novos requisitos, em relação a outras estratégias. Um exemplo de árvore de detecção de eventos pode ser visto na Figura 4. Neste caso, as folhas são referentes aos *wrappers* da fonte GPS, da fonte XML contendo dados da pressão dos pacientes e da fonte relacional com o prontuário dos pacientes.

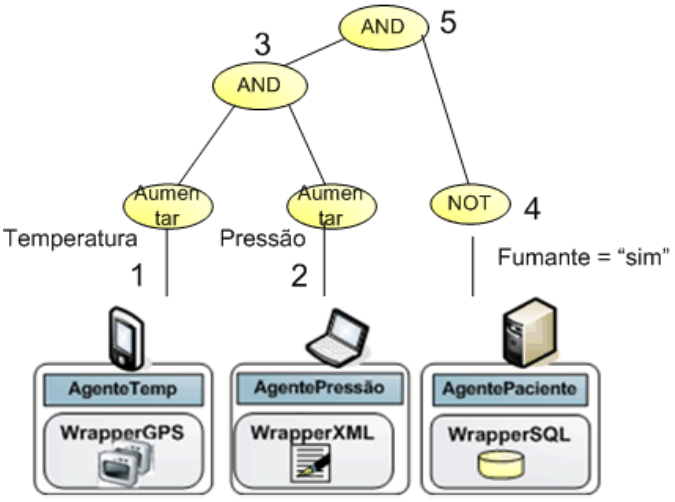

Figura 4: Árvore de eventos

O paralelismo na detecção de eventos pode ser obtido de três maneiras:

- *inter-tree paralelism*: cada evento registado é detectado em um processo separado;
- *inter-operador*: cada operador é um processo separado;
- *intra-operador*: cada tupla é monitorada em um processo em separado. Pode correr naturalmente em fragmentações horizontais.

Cada evento simples, ou operador sendo executado, pode ser considerado uma subtransação. Portanto, subtransações no mesmo nível podem ser executadas concorrentemente.

Eventos representados em XML e lidos pela aplicação através de classes tradutoras têm sido encontrados com freqüência na literatura [BERNAUER, 2004] [PIETZUCH, 2004]. Ele se difere das demais tecnologias devido a fatores como:

- Eventos em XML são ordenados hierarquicamente;
- É uma linguagem rica semanticamente;
- Um *schema* XML pode limitar o número de elementos e atributos em documentos, ou seja, sua cardinalidade. Isto é útil para modelagem, por exemplo, na identificação da presença de três ou mais pessoas em uma determinada sala;
- Tipos de eventos também podem ser hierárquicos e relacionados entre si. O XML é bastante expressivo para esta finalidade, o que possibilita maior reuso destes tipos.

O potencial dos bancos de dados ativos para automatizar ações serve ao objetivo central da computação sensível ao contexto, justificando o seu uso. No caso específico da linguagem XML, é uma linguagem comprovadamente eficiente para representar eventos e realizar a comunicação das partes envolvidas em um SGBDA.

2.3.4 Bancos de Dados Temporais e Espaciais

Os modelos de dados tradicionais apresentam duas dimensões, as linhas (instâncias dos dados) e as colunas (atributos) de uma tabela. Cada atributo de uma instância apresenta um só valor. Se for feita uma alteração deste valor, o anterior é perdido. Bancos de dados temporais permitem armazenar e recuperar todos os estados de uma aplicação (passado e atual), registrando sua evolução com o passar do tempo [EDELWEISS, 1998].

Os modelos temporais, por sua vez, acrescentam mais uma dimensão aos modelos tradicionais: a dimensão temporal. Essa dimensão associa alguma informação temporal a cada valor. Novos valores são acrescentados, sem que os anteriores sejam removidos do banco de dados. Por exemplo, em um atributo que representa o salário de um funcionário, todos os

valores ficam armazenados, com o intervalo nos quais foram válidos. Desse modo, é possível acessar toda a história dos atributos, sendo possível analisar sua evolução temporal. Se visualizarmos as situações passadas como mais uma fonte de informação de contexto, o uso de conceitos de bancos de dados temporais se torna importante nesta área.

Os bancos de dados espaciais, por sua vez, são aqueles preocupados com a localização de entidades e dispositivos, estáticos ou em movimento [VOLZ; SESTER, 2000]. Essas localizações podem ser representadas a partir de coordenadas, ou simbolicamente em alto nível. Um diferencial em relação a pares de atributo-valor comuns é que algum processamento precisa ser feito em certos operadores, como dizer se um objeto se encontra dentro de um determinado cômodo. Normalmente, são criadas sintaxes para a representação do espaço, o que auxilia o uso do espaço como mais um tipo de contexto.

2.3.5 Bancos de Dados Móveis

Um sistema de Bancos de Dados Móveis é aquele cuja localização varia e o acesso à base de dados é feito mediante rede sem fio [OZSU; VALDIUREZ, 2001]. Um sistema em um ambiente móvel está sujeito a falhas tanto de *hardware* como de *software*, como a desconexão quando se sai da área de alcance da rede sem fio. Em cada um desses casos, informações que se referem ao sistema de banco de dados podem ser perdidas. Uma parte essencial do sistema de banco de dados é um esquema de recuperação responsável pela detecção de falhas e pela restauração do banco de dados para um estado consistente que existia antes da falha.

Um dos problemas da portabilidade é o fornecimento de energia: as baterias representam grandes problemas, devido ao seu peso, ao seu tamanho e a sua necessidade de recarga. O consumo de energia depende da capacitância, da voltagem e freqüência do *clock.*

Outro fator que deve ser analisado com cuidado quando se trabalha com bancos de dados móveis é a alta variação da largura da banda, que pode gerar problemas em programas que dependem do envio e recebimento de dados constantes. Além disso, o número de redes heterogêneas exige uma mudança grande de protocolos e de velocidade de transmissão.

A segurança em um ambiente móvel é bem mais complexa devido ao próprio meio aéreo por onde a informação trafega, desprotegido contra tentativas não-autorizadas de acesso, o que vêm originando grande esforço para a solução deste problema.

A gerência de bancos de dados móveis deve considerar várias questões de gerenciamento de metadados, como por exemplo, a atualização constante da localização da informação. Fontes de dados móveis em sistemas sensíveis ao contexto podem ser ambulâncias, turistas em movimento, entre outras informações relevantes.

2.3.6 Bancos de Dados de Tempo Real (RTDBS)

Bancos de dados em tempo real são aqueles onde não se pode haver atraso na entrega dos dados, ou, se inevitável, os atrasos devem ser minimizados. A maioria das abordagens se baseia em atribuições de prioridade e prazos para o término (*deadlines*). [ABBOTT; GARCIA-MOLINA, 1988].

Noções de Sistemas Operacionais podem ser utilizados para este tipo de problema. Um gerenciador é responsável por enviar tarefas, assim como monitorar a fila de processos prontos e em espera, através de um algoritmo de escalonamento (*scheduling algorithm*). O escalonador contém as seguintes informações: tempo de CPU utilizada, limites de espera, informação de estado de *I/O*, lista de dispositivos alocados ao processo e arquivos abertos.

Sistemas não-preemptivos são aqueles nos quais os processos iniciados não podem ser interrompidos em seu *quantum*. Sistemas preemptivos por sua vez são aqueles que um processo recém-chegado de alta prioridade pode ganhar a CPU em detrimento do processo atualmente em execução. Por esta razão, são mais apropriados para sistemas em tempo real, apesar de ser menos eficiente e menos previsível.

O algoritmo *Round Robin* possui uma boa vazão ao dividir processamento entre os diversos processos. Uma outra solução a ser empregada é a utilização de múltiplas filas de escalonamento. Estes algoritmos possuem as seguintes características: (*i*) Os processos são classificados em grupos; (*ii*) Existe uma fila para cada grupo; (*iii*) Cada fila pode ter seu próprio algoritmo.

[ABBOTT; GARCIA-MOLINA, 1988] aborda a questão de *deadlines* para as transações. As informações necessárias para uma previsão sobre o término das transações são: *timestamp* de criação da transação, prazo e porcentagem completa das transações. Quando uma transação não é completa em seu prazo, ela pode ser descartada, ou continuar sendo executada (sendo completa eventualmente). Transações atrasadas podem receber cada vez maior prioridade à medida que o seu atraso aumenta. A Tabela 1 resume os diversos tipos de políticas de prioridade.

Tabela 1: Sumário das políticas de escalonamento (extraído de [ABBOTT; GARCIA-MOLINA, 1988]).

Eligibility	AE - All Eligible NT - Not Tardy FD - Feasible Deadlines
Priority	FCFS - First Come First Serve ED - Earliest Deadline LS - Least Slack
Concurrency	SE - Serial Execution HP - High Priority CR - Conditional Restart

[OZSOYOGLU; SNODGRASS, 1995] realiza um estudo dos diversos gerenciadores de banco de dados em tempo real. [JUDD; STEENISTE, 2003] apresenta uma arquitetura para RTBDS, que possui uma linguagem de consulta semelhante ao SQL, sem demandar, entretanto, que os dados estejam centralizados em um SGBD. Esta linguagem, convertida em XML, é transportada via HTTP, uma alternativa que é comumente encontrada, e será utilizada neste trabalho.

A arquitetura apresentada em [MCFADDEN *et. al.*, 2004] realiza a remoção de subscrições quando estas são completas ou canceladas. Sua escolha para comunicação foi baseada o critério da interoperabilidade, o que elimina opções como RMI.

[LAM, 2000] menciona que o tempo necessário para terminar uma operação é imprevisível. Algumas abordagens anteriores também possibilitavam *deadlock*. Eles definem a prioridade de uma transação como uma função entre seu prazo e criticidade. É recomendável, de acordo com os autores, que as transações disparadas por uma regra sejam uma transação distinta, a fim de não perder seu *deadline*.

Os conceitos de bancos de dados em tempo real podem auxiliar nas aplicações sensíveis ao contexto mais críticas. Entretanto, deve-se ter em mente o aumento da complexidade para a implementação tais sistemas.

2.3.7 Execução Distribuída de Consultas

Em um sistema de banco de dados distribuído, uma Máquina de Execução de Consulta (MEC) possui da função de obter conjuntos resultado parciais em cada fonte de dados e processá-los formando um único conjunto resultado que será enviado à aplicação final. Os operadores de uma MEC Relacional são baseados na álgebra relacional. Porém, as operações algébricas podem ser realizadas de várias formas, devido a diferentes algoritmos. Assim, podemos criar dois conjuntos de operadores algébricos: lógicos e físicos.

Os operadores lógicos são comuns a todas as MECs pois são operadores abstratos, sendo eles: seleção, projeção, produto cartesiano, junção etc. Os operadores físicos são os operadores que a MEC realmente implementa e que os otimizadores de consulta podem utilizar. Os operadores lógicos são mapeados em um ou mais operadores físicos. Por exemplo, o operador lógico junção natural (*natural join*) pode ser implementado como uma junção de laço aninhado (*loop nested join*) ou uma junção união (*merge join*) [AYRES *et. al.*, 2003].

O paralelismo (*pipeline)* do processamento dos resultados no operador de acesso é definido pela forma como as fontes de dados repassarão os resultados à MEC. O operador de acesso pode funcionar de duas maneiras: (*i*) módulo *wait-all*, onde as fontes de dado enviam somente o resultado completo, ou (*ii*) módulo *wait*, onde as fontes de dados enviam partes do resultado, à medida que são criadas. O paralelismo também pode ser obtido em ambientes multi-usuário pelo desacoplamento entre MEC e o processador de consultas. A Tabela 2 mostra alguns exemplos de paradigmas para execução de consultas.

Tabela 2: Características de modelos de execução de consulta (extraído de [AYRES, 2003]).

Características de Execução	*Módulos*	Modelos			
		Tradicional	Adaptável	Contínuo	Streams
Sincronismo	*Wait*	*	*		X
	Wait-all		X		X
	Nowait	X	X	*	*
Distribuição	*Remote*	X	X	X	X
	Local	*	*	X	X
Paralelismo	*Inter*	*	*	X	X
	Intra	X	X	X	*
Fluxo de Dados	*Fixed*	*		X	*
	Adaptive		*	*	X
Fluxo de Controle	*Demand-driven*	*	*	X	X
	Data-driven		X	*	*
Tempo de Resposta	*First-tuple*	*	*	*	*
	Last-tuple		X		X

Legenda: (*)=configuração mais relevante, (X)=configuração possível

Através da criação de várias MECs, várias consultas podem ser atendidas simultaneamente no ambiente, desde que elas se encontrem em nós distintos.

A MEC inserida no CoDIMS foi baseada no *framework* QEEF [AYRES, 2003] e implementada como um componente. Este componente, por ser um *framework*, possui partes fixas, que devem existir em todas as instâncias da MEC, e partes que serão instanciadas dependendo do cenário em que o CoDIMS for utilizado. As partes fixas criam uma infra-estrutura básica para a execução das consultas. Essas partes são a interface da MEC, o processamento do PEC, a estrutura básica dos operadores e do conjunto resultado. As partes instanciadas, responsáveis pela adaptação do CoDIMS a cada cenário, são compostas pelos operadores algébricos e pelo conjunto resultado de cada modelo de dados [PINHEIRO, 2004].

2.4 Tecnologias

Além dos conceitos apresentados nas seções anteriores, algumas linguagens e tecnologias devem ser mencionadas, motivadas por seu uso na implementação da arquitetura proposta citadas em outros projetos relacionados.

2.4.1 XML

Com a criação de *tags*, faz-se necessário definir regras que ajudarão a criar documentos válidos. Validação é o processo de comparação de um documento XML (ou parte de um documento XML) com uma gramática, ou com um conjunto de regras, com a finalidade de determinar se a estrutura do documento está de acordo com as determinações gramaticais e as regras estabelecidas. Uma Definição de Tipo de Documento (DTD) ou um XML *Schema* validam a estrutura de um documento. Pode-se definir, por exemplo, os elementos que se deseja usar no documento e indicar em que ordem estrutural eles deverão ocorrer, se esse elemento pode ser vazio ou deve conter texto. São definidos também, os atributos que devem ser associados com um elemento e cardinalidades. Por estas razões, o XML foi utilizado amplamente na implementação deste trabalho.

2.4.2 CC/PP

O CC/PP (*Composite Capabilities/Preferences Profile*) [INDULSKA *et. al.*, 2003] foi utilizado na definição dos metadados neste trabalho, e é mostrado na Figura 5. Ele é uma recomendação do W3C para a representação da capacidade de dispositivos e preferências de usuários. A seguir é vista a representação de um browser, contendo sua versão e o sistema operacional que o suporta.

```
<rdf:Description rdf:about="http://example.com/Browser">
        <rdf:type rdf:resource="http://example.com/Schema#BrowserUA" />
        <ccpp:defaults>
          <rdf:Description rdf:about="http://example.com/UADefault">
            <rdf:type rdf:resource="http://example.com/Schema#BrowserUA"/>
            <prf:name>Mozilla</prf:name>
            <prf:vendor>Symbian</prf:vendor>
            <prf:version>5.0</prf:version>
</rdf:Description>
```

Figura 5: Exemplo de CC/PP

2.4.3 UserML

O UserML [HECKMANN, 2002] tem como objetivo modelar perfis de usuário para ambientes descentralizados através de triplas RDF. Por este motivo, é baseado em RDF, que também fornece modularização para o modelo de usuário.

Documentos UserML são embutidos em envelopes SOAP. O conteúdo é dividido em cinco partes: metadados, modelo de usuário, explicações de inferências, modelo de contexto, modelo de ambiente. A seguir, na Figura 6, é mostrado um DTD para ilustrar o elemento "explicação para inferência":

```
<!ELEMENT inference (inferred, inferred-from, inferred-by)>
<!ELEMENT inferred-from (evidence)*>
<!ELEMENT evidence (UserData | ContextData)+>
<!ELEMENT inferred-by (device)*>
```

Figura 6: Exemplo de documento UserML

2.4.4 Gumo

O Gumo (*General user model ontology*) [GUMO, 2006] é uma ontologia que pretende fornecer uma visão homogênea para modelos de usuário, mesmo que se encontrem distribuídos, além de lidar com aspectos de segurança de tais informações. Ela utiliza a linguagem OWL, por ser recente e ter expressividade adequada.

O atributo *gumo:expiry*, por exemplo, fornece um valor padrão para o tempo de vida da informação, fora da qual ela é considerada inválida. Um fator a ser considerado são os distintos tempos de vida para tipos distintos de contexto. Por exemplo, podemos ter:

- statusMédico.batimentosCardiacos – pode variar em segundos;
- habilidades.inventivo – pode mudar em meses;
- personalidade.introvertido – leva anos para se modificar;
- demografico.localNascimento – não é modificado.

2.4.5 SensorML

O SensorML [SensorML, 2006] consiste em representações para sensores, tais como termômetros, barômetros, dentre outros, podendo ser incorporado futuramente em alguma aplicação que faça uso de tais sensores. Para todos os sensores, são representados os metadados gerais de cada sensor, cujos elementos são os mesmos para todos os tipos. Depois, o *hardware* é descrito e também as suas entradas, saídas e curvas de erro. A Figura 7 mostra parte do esquema para representar um termômetro, com a sua saída (uma temperatura representada em *Celsius)*.

```
<input name="temperature">
<swe:Quantity definition="urn:ogc:def:phenomenon:temperature"
uom="urn:ogc:def:unit:celsius"/>
</input>
<output name="measuredTemperature">
<swe:Quantity definition="urn:ogc:def:phenomenon:temperature"
uom="urn:ogc:def:unit:celsius"/>
</output>
```

Figura 7: Exemplo de SensorML

2.4.6 ECGML

A *Eletrocardiogram Markup Language* [ECG, 2007] é usada em exames eletrocardiogramas, onde 12 eletrodos são colocados em vários pontos do corpo humano, para uma percepção total do coração no momento. Há mudanças, entretanto, no exame realizado remotamente, onde o paciente se locomove em seu domicílio [ERFIANTO, 2004].

Alguns exemplos de atributos modelados pela linguagem são: a pressão *systolic*, que é a maior pressão, e ocorre no momento que o coração está bombeando o sangue para o corpo; pressão *diastolic* ocorre durante o descanso do coração entre duas batidas.

2.4.7 UML-F

A UML-F é uma linguagem para a modelagem de *frameworks*. Em [FONTOURA *et. al.*, 2000], algumas estruturas criadas são:

- {appl-class} Classes que só existem em instâncias do *framework*, sendo definidas no momento da instanciação;
- {variable} Para métodos. O método deve ser implementado durante a instanciação;
- {extensible} A interface da classe depende da instância, novos métodos podem ser definidos para estendê-la;
- {static} A informação necessária é dada em tempo de compilação;
- {dynamic} Informação faltando dada em tempo de execução;
- {incomplete} Novas subclasses podem ser adicionadas.

A seleção da estratégia mais apropriada é uma tarefa criativa, necessita de intervenção humana. Algumas ferramentas, como o Rational Rose, permitem a customização da geração automática de código.

Descrições UML-F podem ser vistas como "receitas" que dizem onde códigos específicos devem ser acrescentados. Existe um tutor para a criação de instâncias. Ela identifica todos os pontos onde código deve ser fornecido. Nos diagramas UML-F, tem-se como destaque os *hotspots*, além das restrições pra instanciação.

A MEC original do CoDIMS foi escrita originalmente em UML-F [PINHEIRO, 2004]. Este trabalho também se utiliza desta linguagem para especificação de componentes.

2.4.8 SOAP

O SOAP [W3SCHOOLS, 2006] é um protocolo definido pelo W3C para o intercâmbio de mensagens entre *Web Services*. As mensagens SOAP obedecem a uma sintaxe própria baseada em XML, e que utiliza o HTTP como protocolo de transporte, o que evita alguns problemas de acesso e devido à *firewalls*. As principais partes de uma mensagem SOAP são:

- Envelope: onde se encontram algumas informações tais como o estilo de codificação;
- Cabeçalho: Fornece informações sobre o transporte na rede, entre outras;
- Corpo: é a mensagem propriamente dita, onde também se pode armazenar *logs* de falhas que porventura ocorram.

Um exemplo de mensagem SOAP é para o sistema de vendas visto na Figura 8. Neste exemplo, um usuário solicita detalhes de um produto através do *id* do produto. A mensagem de retorno, fornecida pelo *web service*, retorna as informações do produto.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetails xmlns="http://warehouse.example.com/ws">
       <produtId>827635</produtId>
     </getProductDetails>
   </soap:Body>
 </soap:Envelope>
```

Figura 8: Exemplo de mensagem SOAP

3 Requisitos para Integração de Dados Contextuais

Neste capítulo são apresentados os requisitos funcionais e não funcionais identificados para a construção de uma arquitetura de acesso, integração e manipulação de dados contextuais. O levantamento e análise de tais requisitos são necessários para a especificação da arquitetura de acesso e integração de dados contextuais proposta nesse trabalho.

Em [KAPSAMMER *et. al.* 2005] são apresentados, de maneira simplificada, alguns requisitos funcionais relativos à manipulação de dados contextuais. Durante a fase de projeto e definição dos componentes funcionais da arquitetura proposta, outros requisitos foram identificados como importantes e necessários, sendo apresentados a seguir.

3.1 Requisitos Primários

Esta seção abrange os requisitos abordados em [KAPSAMMER *et. al.*, 2005].

3.1.1 Acesso e Integração de Dados Contextuais

Nas aplicações *desktop* tradicionais freqüentemente devem ser integrados dados de fontes de natureza heterogênea ou de sistemas legados pertencentes a organizações distintas. Sistemas sensíveis ao contexto lidam adicionalmente com dados provenientes de sensores e dispositivos de aquisição diversos, com tecnologias de comunicação e transmissão, configurações e capacidades distintas. Isso torna o problema de integração potencialmente mais complexo, impondo à arquitetura a definição de componentes funcionais voltados para tal propósito. Em muitos projetos, os provedores de contexto são encapsulados e acessados através de uma mesma interface de acesso. Para projetos onde o acesso é centralizado em um componente, há a necessidade de um esquema global para consulta e interface com sistemas externos. Para tanto, deve-se realizar a tradução entre os conjuntos resultados obtidos e os formatos utilizado pelas aplicações requisitantes, que podem ser SOAP, OWL, entre outros.

Ainda há a questão de novas aplicações e evolução de *software*. É reconhecida [PASCOE, 2001] a impossibilidade de se modelar todos os domínios para uma aplicação, o que torna mandatória a adição de novos modelos e tipos de informação.

Existem algumas iniciativas proprietárias de integração de dados focadas em dispositivos móveis, como o Oracle Lite e o DB2 *EveryPlace*. Entretanto, são soluções pontuais para determinados SGBDs, acoplando a solução a tecnologias específicas. Em uma arquitetura de propósito geral, existem outros dispositivos que devem ser englobados. A Tabela 3 mostra os casos de uso para inserção de fontes, dados e subscrições.

Tabela 3: Requisito Acesso e Integração de dados

Atores	Usuário final
Casos de uso associados	Inserção de fonte de dados Inserção de dados Inserção de subscrição
Requisitos dependentes	
Requisitos dos quais depende	Metadados

3.1.2 Dinamismo das Informações

No ambiente sensível ao contexto, os dados são altamente dinâmicos, onde características como a validade temporal devem ser avaliadas a todo instante. Devem-se incorporar mecanismos para garantir a consistência e a atualidade das informações contextuais provenientes das fontes, a fim de se evitar o uso de dados inválidos e/ou desatualizados pela aplicação. No caso de indisponibilidade dos dados mais atuais, uma estratégia é atribuir diferentes pesos a cada fonte de dados de acordo com grau de atualização das mesmas. No domínio específico de sistemas geográficos, por exemplo, atenção especial deve ser dada às regiões de fronteira, pois mais de um sensor pode captar a mesma informação. Pode-se ver que o mesmo ocorre durante trocas de contexto quando são utilizados dispositivos móveis [CELENTANO *et. al.*, 2004]. A Tabela 4 mostra como este requisito não funcional impacta na entrega de dados.

Tabela 4: Requisito dinamismo nas informações

Atores	Usuário final
Casos de uso associados	
Requisitos dependentes	Entrega ativa de dados
Requisitos dos quais depende	

3.1.3 Metadados

Formas flexíveis e expressivas para a descrição de metadados, como o uso de ontologias, devem estar presentes para que os dados contextuais sejam facilmente adicionados e

recuperados, não somente pelas aplicações, mas também pelos demais componentes dos sistemas de integração de dados contextuais. Além disso, metadados podem descrever resultados de operações, tais como consulta realizada com sucesso, fonte não conectada, *timeout*, fonte não localizada, dentre outros [CÔCO, 2004]. Esta funcionalidade é muito importante em nosso ambiente com dados sensoriais, pois, como será visto, sensores e fontes móveis apresentam freqüentes problemas de conectividade. Finalmente, os metadados podem descrever os eventos monitorados. [BUNNINGEN; APERS, 2005] reafirma a necessidade de uso dos metadados.

Tabela 5: Requisito Metadados

Atores	Usuário final
Casos de uso associados	Inserção de metadados Remoção de metadados
Requisitos dependentes	Integração de dados Entrega ativa de dados Adição dinâmica de fontes Dinamismo das informações
Requisitos dos quais depende	

Os metadados, em especial aqueles descritos por ontologias, podem auxiliar em diversos requisitos mencionados anteriormente:

a) Integração de dados heterogêneos. Ontologias auxiliam na descrição das relações e atributos em fontes, além de aspectos específicos de sensores, como precisão. Tais descrições são importantes internamente para a elaboração de planos de consulta, e também às aplicações requisitantes, como interpretadores de contexto. Metadados sobre a autoria de certas fontes também podem ser de grande utilidade.

b) Entrega ativa de dados: As ontologias auxiliam no envio ativo de informações, no sentido que as descrições das fontes se constituem em restrições de integridade sobre seu envio. Por exemplo, o sistema deve mencionar ao usuário se ele tentar receber dados de um sensor com uma taxa de atualização maior do que a efetivamente suportada pelo mesmo.

c) Dinamismo das informações: Em sistemas sensíveis ao contexto, portanto, os metadados se tornam tão importantes quanto os dados em si, sendo vitais para a confiabilidade do sistema. Os metadados são em si fontes de dados a serem acessadas para caracterização do ambiente. Um interpretador de contexto pode, por exemplo, atribuir diferentes pesos às diversas informações, não comprometendo assim seu poder de inferência.

d) Histórico contextual: a obtenção de dados relevantes historicamente pode se beneficiar de informações como a data inicial em que uma fonte de dados foi disponibilizada.

e) Problemas de mobilidade e miniaturização de dispositivos. Ontologias podem auxiliar heurísticas que se baseiem em acessar as fontes de maior confiabilidade, maior taxa de atualização ou maior energia restante, para não se interromper a consulta prematuramente.

A figura 9 mostra um mapeamento entre atributos de pacientes, que possibilita o uso de duas fontes que usam diferentes modelos de dados, contendo atributos em comum entre elas através do uso de um esquema global.

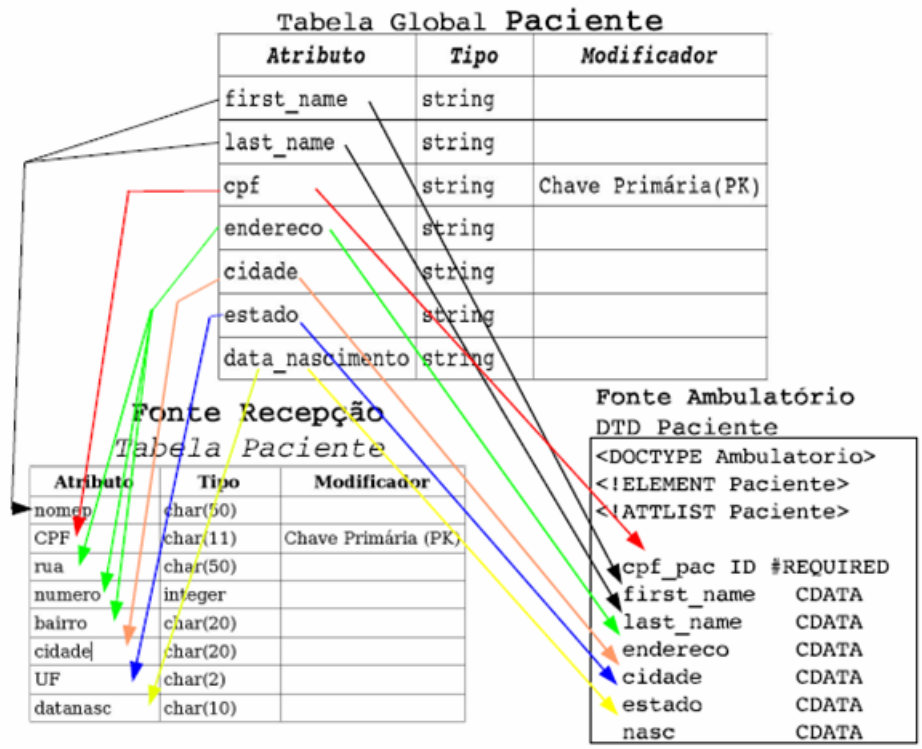

Figura 9: Mapeamento de metadados (extraído de [SILVESTRE, 2005]).

3.1.4 Perfil de Usuário

Os perfis dos usuários são utilizados na personalização do acesso aos dados, ou seja, os dados que são solicitados pelas aplicações devem ser retornados de acordo com as características e desejos e importância para os usuários. As informações obtidas das fontes devem ser selecionadas em função da vontade do usuário, expressa em seu perfil. Alguns exemplos de informações sobre o usuário são: idioma nativo, cidade, nível acadêmico, bagagem cultural, dentre outras. Estas informações são aplicadas como um filtro à consulta, e idealmente devem

diminuir a cardinalidade do conjunto resultado. Os fatores citados acima adicionam complexidade à otimização distribuída de consulta, onde se deve preocupar com a ordem de execução dos operadores.

Os tipos de informação de perfil podem ser valores quantitativos, que são comparações diretas com o contexto do usuário, por exemplo, a sua localização. As informações de perfil também podem ser atributos que devem ser fornecidos pelo usuário, como autores de livros favoritos. As preferências das pessoas não são sempre estáticas, mas dependem do contexto atual, das situações em que se encontram. Por exemplo, um usuário pode especificar que quer ouvir músicas lentas durante o serviço, e programas de notícia durante as refeições [VASTENBURG, 2004].

O usuário deve saber, quando necessário, o porquê uma decisão foi tomada pelo sistema. Idealmente ela deve ser descrita em alto nível e o usuário deve ter o poder de cancelar as ações decididas. O contexto utilizado para tais decisões deve ser tratado da mesma maneira que uma entrada explícita [BUNNINGEN; APERS, 2005].

Tabela 6: Requisito Perfil de usuário

Atores	Usuário final
Casos de uso associados	Inserção de perfil Remoção de perfil Modificação de perfil
Requisitos dependentes	
Requisitos dos quais depende	Integração de dados

3.1.5 Contexto Espaço-tempo-computacional

A forma de acesso e integração de dados deve incluir abordagens diferenciadas no que se refere à localização espacial e temporal, de forma que os mesmos possam refletir exatamente a informação desejada. Estas informações são utilizadas em consultas do tipo *location-aware* [DEY, 2000], ou seja, consultas baseadas na localização do usuário, vitais em um ambiente sensível a contexto. Não devem ser ignorados os recursos computacionais existentes nas fontes para o processamento destes dados.

Tabela 7: Requisito contexto espaço-tempo-computacional

Atores	Usuário final
Casos de uso associados	
Requisitos dependentes	Linguagem temporal Linguagem espacial
Requisitos dos quais depende	Integração de dados

3.2 Requisitos Adicionais

Durante a fase de projeto e definição dos componentes funcionais da arquitetura proposta, outros requisitos foram identificados como necessários, sendo apresentados a seguir.

3.2.1 Configuração e Escalabilidade

Considerando o dinamismo das aplicações sensíveis ao contexto, os requisitos funcionais associados a cada uma delas podem se tornar obsoletos rapidamente, havendo a necessidade de incorporação de novas funcionalidades. Assim, as arquiteturas criadas para atender a estas aplicações devem se basear em técnicas e metodologias de desenvolvimento que favoreçam reutilização, adaptabilidade e extensibilidade. Esta estratégia visa atender mais facilmente o desenvolvimento das aplicações e a alteração e/ou incorporação de novos requisitos.

No início da computação sensível ao contexto, muito esforço foi dispendido em aplicações particulares, sem a preocupação com requisitos como modularidade, reusabilidade e interoperabilidade. Após este estágio, uma abordagem de *framework* foi iniciada, sendo o exemplo mais famoso o *Context Toolkit* [DEY, 2000].

Em [GRISWOLD *et. al.*, 2003], um estudo sobre extensibilidade em sistemas sensíveis ao contexto foi realizado, identificando os seguintes pontos desejados: novos dispositivos, novos tipos e formatos de dados, novas aplicações finais.

A Tabela 8 mostra que a configuração é um dos requisitos que mais influem em outros requisitos, sendo crucial para o sucesso de plataformas.

Tabela 8: Requisito Configuração e escalabilidade

Atores	Usuário final
Casos de uso associados	Inserção de componente Remoção de componente
Requisitos dependentes	Metadados Perfil de usuário Histórico contextual Aspecto espacial Entrega ativa
Requisitos dos quais depende	

3.2.2 Entrega Ativa de Dados

Uma das principais características da computação sensível ao contexto é a entrega ativa de informações relevantes ao usuário que pode ser basicamente realizada de três maneiras distintas: **(i)** *Request-response*: requisições *ad hoc*, ou seja, realizadas pelo usuário quando necessário; **(ii)** periodicamente (*time-driven*): requisições realizadas em intervalos de tempo pré-definidos; e **(iii)** baseada em evento (*event-driven*): entrega dos dados quando da ocorrência de algum fato relevante. Tal funcionalidade pode ser atendida através do uso dos conceitos de bancos de dados ativos, com o uso de regras e de mecanismos disparadores de ações de acordo com as necessidades da aplicação. Deve-se ter a funcionalidade de monitorar várias condições simultaneamente, possibilitando a construção de eventos compostos que enriquecem o ambiente.

[VARGAS *et. al.*, 2004] descreve os bancos de dados ativos e o paradigma publicação/subscrição. Tal como nos históricos de dados, o nível de abstração pode ser configurado pelo usuário, para se evitar *overhead* de informações [POKRAEV *et. al.*, 2003].

Finalmente, [MEIER; CAHILL, 2005] apresentam uma taxonomia para sistemas baseados em eventos, abordando aspectos como distribuição, modos de entrega, expressividade da linguagem, presença de eventos compostos e qualidade de serviço.

Tabela 9: Requisito Configuração e escalabilidade

Atores	Usuário final, administrador
Casos de uso associados	Inserção de subscrição
Requisitos dependentes	Linguagem de eventos
Requisitos dos quais depende	Metadados Histórico contextual Prioridade entre consultas

3.2.3 Modificações nas Regras ACID

Um dos maiores problemas para o gerenciamento de transações distribuídas ocorre quando existem problemas de conectividade e tempo de espera, podendo tornar a teoria de seriabilidade inapropriada para certas aplicações, como apontado em [ÖZSU; VALDURIEZ 2001]. Tal fato pode acarretar o armazenamento de dados inconsistentes e o não sincronismo das réplicas existentes, gerando informações contextuais inválidas ou duvidosas. Processadores de consultas mais robustos devem ser projetados e utilizados no caso de atualização e uso de réplicas, principalmente em dispositivos móveis.

Controle de concorrência: No caso de aplicações *groupware*, há a necessidade de um controle de concorrência mais sofisticado no lado servidor, sendo desaconselhados algoritmos pessimistas, pois caso um usuário que esteja de posse de uma permissão perca conexão ocorrerá *deadlock*, o que compromete uma sessão inteira [HALEVY, 2003]

Tabela 10: Requisito Modificações nas regras ACID

Atores	
Casos de uso associados	
Requisitos dependentes	Entrega ativa de dados
Requisitos dos quais depende	

3.2.4 Histórico Contextual

As informações contextuais, muitas vezes, devem ser armazenadas de maneira a serem futuramente consultadas por aplicações interessadas. A evolução histórica de uma informação constitui, ela mesma, uma nova informação de contexto, podendo ser interpretada e utilizada em inferências futuras. Arquiteturas para sistemas sensíveis ao contexto devem prover meios flexíveis para periodicamente extrair, consolidar e armazenar as informações.

Tabela 11: Requisito Histórico contextual

Atores	Usuário final
Casos de uso associados	Inserção de histórico Recuperar histórico
Requisitos dependentes	Entrega ativa de dados
Requisitos dos quais depende	Acesso e integração de dados

3.2.5 Mobilidade e Problemas de Miniaturização

Estes problemas se referem, de uma maneira geral, à largura de banda, fonte de alimentação, capacidade de processamento, armazenamento e conectividade. Segundo [BIERMAN *et. al.*, 2003], torna-se necessário otimizar a quantidade de informações processadas no local onde o dado se encontra e minimizar o tráfego na rede, que é mais complexo em fontes móveis. Heurísticas que se baseiem em fatores como confiabilidade das fontes ou taxa de atualização podem ser utilizadas para estes desafios. Apesar das redes móveis já terem sido exaustivamente estudadas, no uso de sensores torna-se necessário aprofundar os estudos em caso de falhas de medições.

Tabela 12: Requisito Mobilidade

Atores	
Casos de uso associados	
Requisitos dependentes	
Requisitos dos quais depende	Acesso e integração de dados Entrega ativa de dados

3.2.6 Adição Dinâmica de Fonte de Dados

As fontes de dados, o domínio e as aplicações a serem executadas podem não ser conhecidos antecipadamente pelos sistemas sensíveis ao contexto. Dessa forma, não é viável interromper o sistema para reconfigurá-lo a cada vez que uma fonte é acrescentada ou removida. As arquiteturas devem permitir a reconfiguração dinâmica de novas fontes de dados em tempo de execução, considerando a possibilidade de existir grande número de fontes envolvidas.

[REY; COUTAZ, 2004] reforçam a necessidade de infra-estruturas de processamento em tempo de execução: no projeto Contextor, a busca de fontes é realizada em tempo de execução, mas através de recursos de rede. O trabalho encontrado em [BRAYNER; FILHO, 2002] descreve o projeto AMDB, uma plataforma para ambientes móveis configuráveis dinamicamente, através de agentes. Estes são alguns dos exemplos, de abordagens para a adição dinâmica de fontes de dados.

Tabela 13: Adição dinâmica de fontes de dados

Atores	Administrador do sistema
Casos de uso associados	Inserção de fonte Remoção de fonte
Requisitos dependentes	
Requisitos dos quais depende	Metadados

3.2.7 Prioridade entre Consultas

As consultas submetidas devem possuir um critério de prioridade de forma a qualificá-las e alocar recursos de acordo com a necessidade, importância e domínio de aplicação contextual envolvida. Em um ambiente hospitalar, consultas relativas aos sinais vitais de pacientes têm maior prioridade que dados administrativos. Nos ambientes tradicionais de integração de dados, as penalidades normalmente são atrasos na execução: em computação sensível ao contexto, podem definir o sucesso ou não de uma tarefa. As prioridades globais são apontadas por um administrador do sistema [OZSU; VALDURIEZ, 2001].

Um escalonador de consultas é o componente do SGBD que aloca os recursos disponíveis (memória, unidades de discos e processadores) aos nós de uma árvore de

operadores. Além do plano de execução da consulta, o escalonador utiliza informações disponíveis no meta-esquema, ou até em informações obtidas em tempo real sobre a carga de cada nó, para gerar o plano de alocação que determina os nós de processamento nos quais grupos de operadores serão executados. Um sistema é preemptivo, quando na entrada de um processo de maior prioridade, o processo atual é suspenso imediatamente. Sistemas não-preemptivos terminam o processo atual mesmo com a chegada de processo prioritário.

Tabela 14: Prioridade entre consultas

Atores	Administrador do sistema
Casos de uso associados	Atribuir prioridade a subscrição
Requisitos dependentes	Entrega ativa
Requisitos dos quais depende	

3.2.8 Linguagem de Eventos

É importante a descrição detalhada de eventos, visto que na computação sensível ao contexto, os eventos representam abstrações da situação do ambiente do qual o usuário está interessado.

No projeto DBGlobe, foi criado o Active XML, uma extensão do XML, que se trata de arquivos XML com dados comuns, por exemplo a temperatura de uma cidade, obtida a partir de um *web service* de informações climáticas. A linguagem XML é amplamente utilizada por sua ampla aceitação, facilidade de manipulação e interoperabilidade.

O GML (*Geographical Markup Language*) [GML, 2006] representa diversas formas geométricas e topológicas, com várias unidades possíveis. Estas características contribuem para a representação dos eventos de interesse.

Tabela 15: Linguagens de eventos

Atores	Administrador do sistema
Casos de uso associados	Modificar linguagem de eventos
Requisitos dependentes	
Requisitos dos quais depende	Entrega ativa de dados

3.2.9 Linguagem Espaço-temporal

Na computação sensível a contexto, normalmente têm-se um nível de abstração bastante alto, o que muitas vezes se reflete no próprio formato das consultas utilizadas pelas aplicações sensíveis a contexto. Nesse sentido, algumas extensões para SQL foram propostas para lidar com intervalos de tempo e espaço. A plataforma MoGATU [PERICH *et. al.*, 2004], além das

cláusulas convencionais SQL, usa algumas outras cláusulas, como a cláusula “TIME”, responsável pelos aspectos temporais dos dados consultados. Com ela pode-se especificar na consulta a validade temporal dos dados consultados. Ocasionalmente, novas linguagens são criadas para a execução de consultas sobre dados contextuais. A arquitetura Nexus [VOLZ *et. al.*, 2000] define a linguagem AWQL (*Augmented Word Query Language*), apropriada para ambientes sensíveis a contexto.

Tabela 16: Linguagem espaço-temporal

Atores	Administrador do sistema
Casos de uso associados	Modificar linguagem
Requisitos dependentes	
Requisitos dos quais depende	Aspecto espaço-temporal

3.2.10 Segurança e privacidade

Na computação sensível ao contexto, assim como em SGBDs, deve-se garantir confidencialidade dos dados, ou seja, apenas pessoas autorizadas podem ter acesso a uma certa informação, sejam atribuídos indivualmente ou de acordo com os papéis do usuário no ambiente. Isto inclui a segurança das fontes de dados em si e durante a comunicação entre os componentes das plataformas.

Tabela 17: Segurança

Atores	Administrador do sistema
Casos de uso associados	
Requisitos dependentes	Acesso e integração de dados
Requisitos dos quais depende	

4 Trabalhos Relacionados

Neste capítulo serão apontadas as principais arquiteturas para integração de informações contextuais, além de trabalhos relacionados específicos para os requisitos do capítulo 3.

A área de aplicações móveis sensíveis ao contexto ainda não foi devidamente explorada pela comunidade de banco de dados. Especificamente sobre sistemas de integração de dados contextuais, são encontrados poucos projetos na literatura. Porém, nota-se que tais sistemas foram desenvolvidos visando aplicações específicas ou conjuntos restritos de aplicações, o que dificulta o seu uso em outras áreas de aplicações ou a incorporação de novos requisitos.

Inicialmente serão apresentados os projetos Infraware e CoDIMS no intuito de contextualizar as necessidades de integração de dados oriundos de diversos dispositivos provedores de contexto, no caso da Infraware, e as funcionalidades apresentadas pelo CoDIMS para integração de dados. Em seguida, são apresentadas arquiteturas voltadas para aplicações sensíveis ao contexto, com maior similaridade a este trabalho.

4.1 A Plataforma INFRAWARE

A plataforma Infraware é um *middleware* baseado na tecnologia de *web services* com suporte arquitetural para o desenvolvimento, construção e execução de aplicações móveis sensíveis ao contexto, desenvolvido no Laboratório de Pesquisa em Redes e Multimídia (LPRM) do Departamento de Informática da UFES. A arquitetura conceitual da Infraware estende a da plataforma WASP [COSTA, 2003], um projeto holandês desenvolvido pela *University of Twente*, Telematica Instituut e Ericsson, e que posteriormente teve continuidade através de uma parceria institucional entre a *University of Twente* e o Laboratório LPRM.

A Infraware foi definida tendo como intenção o atendimento a um grupo de requisitos levantados e a integração destes em uma infra-estrutura única, formando, assim, um ambiente propício ao desenvolvimento de aplicações ubíquas em domínios variados, um dos objetivos do projeto DBMWare. Inicialmente, as aplicações-alvo são aquelas do domínio da Saúde, em que os conceitos de sensibilidade ao contexto e a necessidade do uso de *middlewares* de suporte a aplicações sensíveis ao contexto são facilmente evidenciados e diretamente aplicados. A Figura 10 ilustra a arquitetura geral da plataforma Infraware.

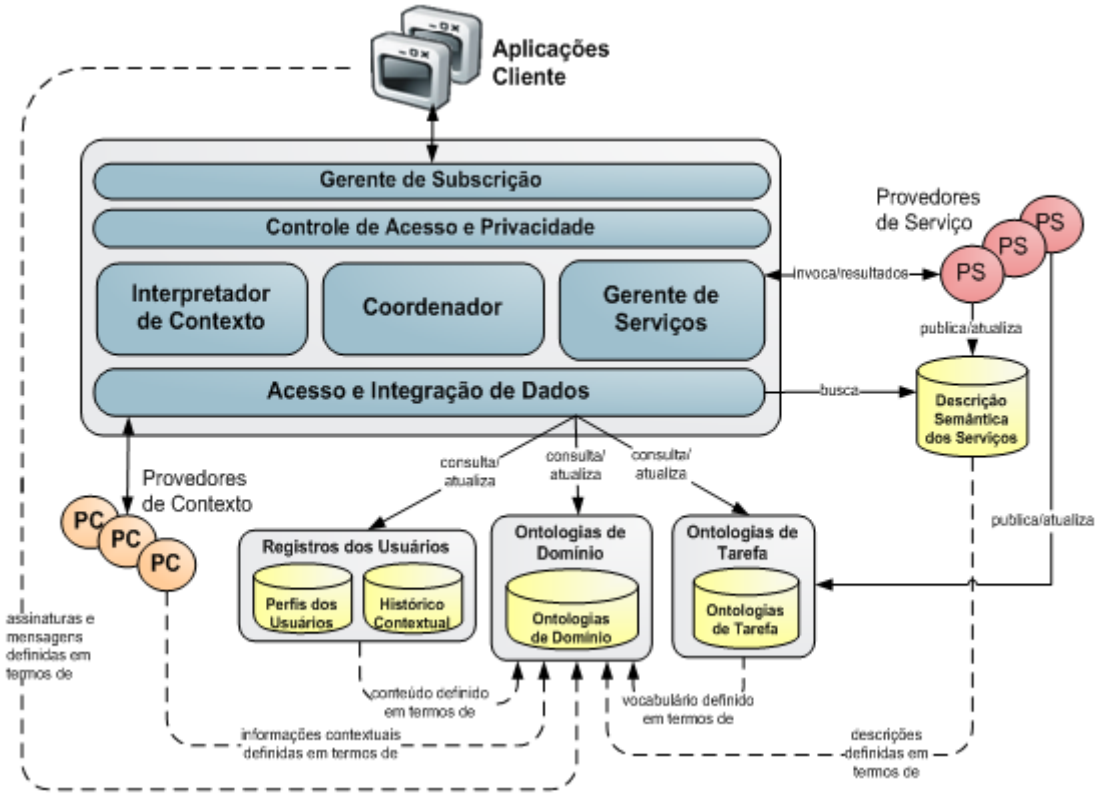

Figura 10: A Plataforma Infraware.

Os objetivos específicos da plataforma Infraware são:

1 - Desenvolver uma plataforma de serviços *context-aware*, baseada em *web services*, para suportar a execução de aplicações móveis sensíveis ao contexto. A arquitetura da plataforma deverá facilitar a concepção dinâmica de novas aplicações, abstraindo a complexidade da manipulação de informações contextuais.

2 – Como prova de conceito, desenvolver uma aplicação móvel piloto, sensível ao contexto, de monitoramento de pacientes em ambiente hospitalar, com suporte para trabalho cooperativo (*whiteboard*, conferência, etc.) usando um sistema de prontuário eletrônico de paciente (PEP) como base de apoio de informações sobre os pacientes. O protótipo deverá ser testado em unidade hospitalar do Governo Estadual e/ou no Hospital Universitário da UFES.

Para ilustrar o fluxo de funcionamento da arquitetura, seja o seguinte exemplo de aplicação facilitada através do uso da plataforma Infraware, mostrado na Figura 11: Um médico responsável pelo paciente com problemas cardiovasculares pode solicitar o monitoramento desse paciente através de uma aplicação sensível ao contexto voltada a tal propósito. Essa aplicação, inicialmente, realiza um pedido de subscrição de serviços à plataforma através da linguagem de subscrição da Infraware e de parâmetros específicos

determinados pelo usuário. O módulo Gerente de Subscrição recebe, decodifica e analisa o pedido recebido, transmitindo seus dados ao componente Controle de Acesso e Privacidade que, através de informações de perfis do paciente, verifica se o nível de acesso requisitado pelo médico é respeitado. Em seguida, o componente Coordenador, a partir dos dados da subscrição e de modelos de privacidade e de visibilidade do usuário, gera um plano de execução de atividades que será disparado aos demais componentes da Infraware, especialmente ao componente Acesso e Integração de Dados, ao Interpretador de Contexto, e ao Gerente de Serviços. Ao componente Acesso e Integração de Dados caberá a obtenção de dados contextuais primitivos, tais como a freqüência cardíaca e a pressão sangüínea, provenientes das diversas fontes de informações existentes, que, nesse caso, são sensores ligados ao paciente capazes de medir seu eletrocardiograma.

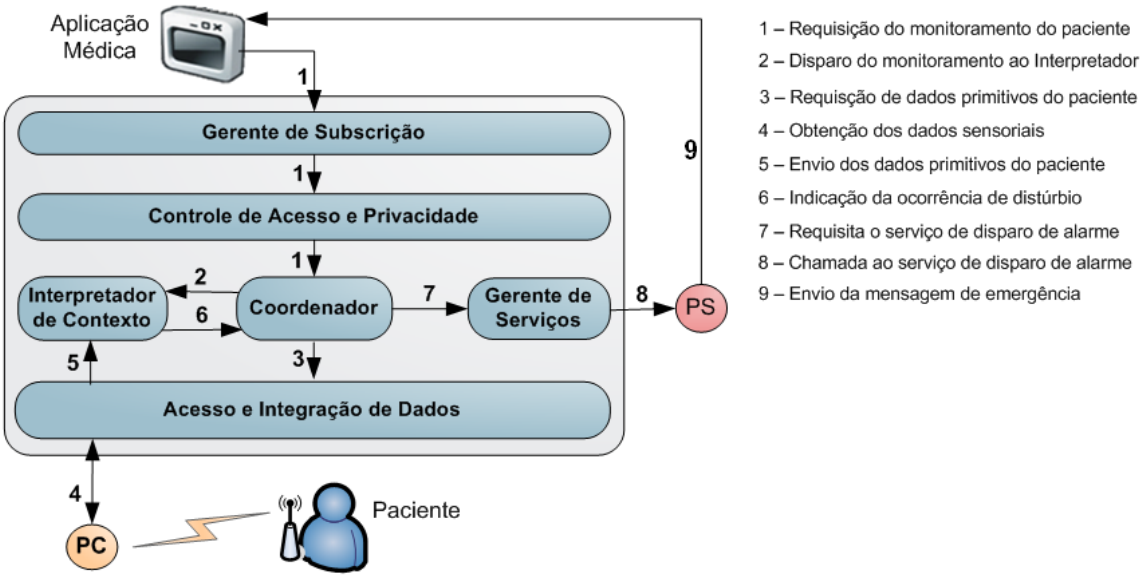

Figura 11: Cenário Médico-hospitalar (extraída de [FILHO et al., 2006

O componente Acesso e Integração de Dados realiza consultas aos provedores de contexto e abstrai para os demais componentes do *middleware* a complexidade da tecnologia utilizada no ECG e em outros dispositivos. As informações contextuais primitivas coletadas pelo componente Acesso e Integração de Dados são entregues ao Interpretador de Contexto, que realiza o monitoramento desses dados e, através de processos de inferências e derivações, gera informações contextuais mais refinadas com o intuito de detectar anomalias ou distúrbios nos batimentos cardíacos do paciente. O módulo Gerente de Serviços irá disparar, quando necessário, serviços de alerta de emergência à aplicação solicitante do monitoramento do paciente (Figura 11(b)). O componente de Descoberta de Serviços do Gerente de Serviços,

através do protocolo SCaSDP, irá buscar e localizar os serviços de alertas disponíveis na ocorrência da emergência. Além disso, o componente de Descoberta de Serviços também auxilia o componente Acesso e Integração de Dados na descoberta dos provedores de contexto que informam os dados cardiológicos do paciente monitorado.

Dentre as necessidades da Infraware, uma delas está diretamente relacionada a este trabalho: a integração de dados oriundos de diversos dispositivos chamados de provedores de contexto. Os provedores de contexto se caracterizam pelo dinamismo, mobilidade, validade temporal, validade geográfica, dentre outros fatores, que os diferenciam das fontes de dados tradicionais. Além disso, devido ao aspecto pró-ativo das aplicações sensíveis ao contexto, os dados devem ser entregues quando eventos subscritos pelo usuário são identificados. Para atender a tal requisito foi então identificada a necessidade de especificação e implementação do componente Acesso e Integração de Dados (Figura 1) para atender aos referidos projetos. Com tal objetivo, foi planejada a adaptação do *framework* CoDIMS [BARBOSA *et. al.*, 2002] como base proposta para o desenvolvimento do módulo Acesso e Integração de Dados.

4.2 CoDIMS

O CoDIMS é um *middleware* para integração de dados heterogêneos e distribuídos utilizando técnicas de *frameworks* e componentes com o objetivo de gerar sistemas de integração de dados configuráveis e flexíveis para aplicações específicas. Os componentes do CoDIMS são implementados como *web services*, permitindo a eles estarem localizados em nós distintos de uma rede [BIANCARDI *et. al.*, 2005]. Novas funcionalidades podem ser acrescentadas implementando-se novos ou reutilizando-se componentes. Outros exemplos de instâncias do CoDIMS são apresentados em [DA SILVA *et. al.*, 2006] e [TREVISOL *et. al.*, 2007].

Os seus principais componentes (apresentados na Figura 12) constituem a configuração mínima, devendo sempre estar presente, e são: (*i*) **Controle**, que gerencia a configuração e execução; (*ii*) **Processamento de Consulta**, que trata de toda a parte de análise, reescrita, otimização e Máquina de Execução de Consulta (MEC); (*iii*) **Metadados**, que contém todas as informações relativas às fontes de dados; e (*iv*) **Acesso a Dados**, responsável por fazer o acesso a cada fonte de dados. Quando uma consulta é enviada, ela é interpretada sendo geradas as sub-consultas que serão encaminhadas ao Acesso a Dados e por sua vez repassadas aos *wrappers*. Cada *wrapper* manipula uma fonte de dados e é responsável

pela execução da sub-consulta a ela dirigida. Cada sub-resultado retorna à MEC, onde são integrados para gerar o resultado final.

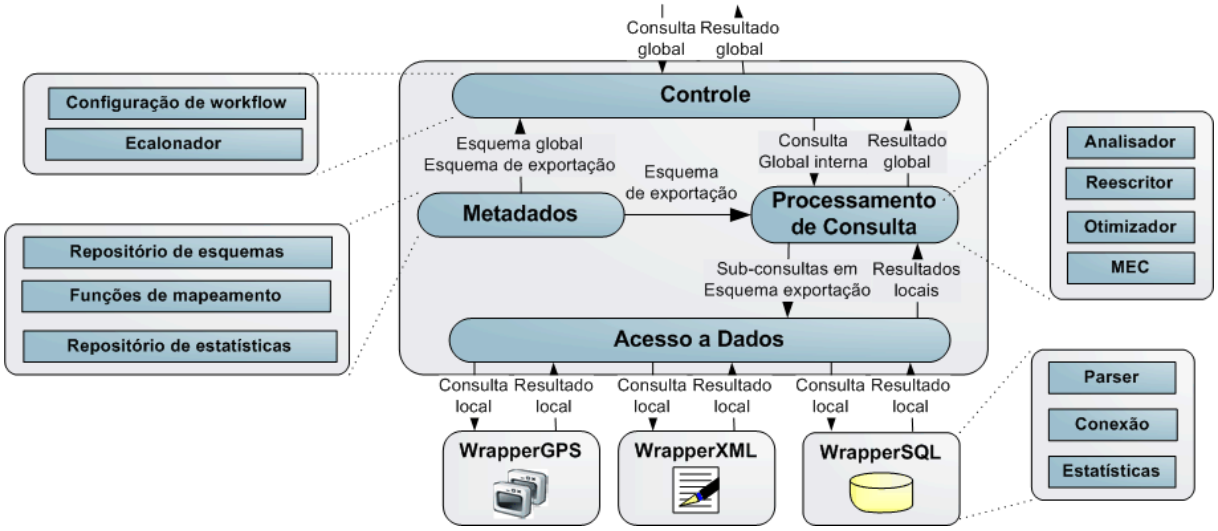

Figura 12: Arquitetura inicial do CoDIMS

Dentre os componentes já complementados, destacam-se o de Metadados [SILVESTRE, 2005], a MEC [PINHEIRO, 2004] e o Otimizador de Consultas [GOMES, 2005], além de *wrappers* [CÔCO, 2005]. Toda a configuração, representação de consultas e resultados são realizadas em XML, permitindo flexibilidade na comunicação dos componentes.

Para facilitar o uso de componentes e *web services*, foram utilizados padrões de análise [GAMMA, 1995], em especial, o padrão *Facade*. O padrão *Facade* fornece, para um conjunto de interfaces de um subsistema, uma interface unificada de alto nível que facilita o uso do subsistema. A estruturação de um sistema em subsistemas auxilia a redução da complexidade. Como um dos objetivos de um projeto é minimizar a comunicação e dependências entre os subsistemas, a introdução de *Facade* fornece uma interface única e simplificada para uma maior facilidade de comunicação com o subsistema. O exemplo de comunicações entre fachadas é visto da Figura 13. As fachadas do Processamento de Consultas e Controle se comunicam, e todas as demais classes apenas se comunicam com classes do mesmo pacote.

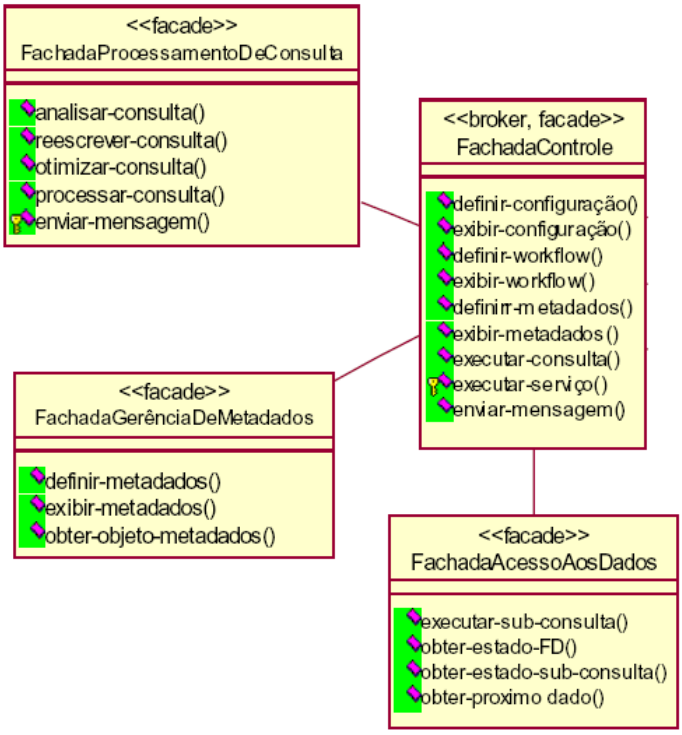

Figura 13: Comunicação entre fachadas do CoDIMS (extraído de [BARBOSA, 2001])

Após um trabalho de levantamento de requisitos para integração de dados contextuais, uma nova instanciação do CODIMS foi proposta para realizar a função de acesso e integração de dados na plataforma Infraware. Seu uso inicial será no escopo da telemedicina. Futuramente, esta instância poderá ser utilizada em outros domínios, o que justifica a reconfiguração como objetivo prioritário. A reutilização e adaptação de seus componentes para diferentes domínios de aplicação constituem os pontos fortes para o seu uso em plataformas de propósito geral, e em particular as para computação sensível ao contexto.

4.3 Nexus

Desenvolvido na *University of Stuttgart*, a plataforma Nexus [BAUER, 2004] tem como componente central a localização de entidades e dados espaciais, provida através de uma infra-estrutura que trata de informações espaciais, por meio de linguagens de consulta, armazenamento de dados e eventos apropriados. O Nexus possui uma camada de federação de dados, que realiza as etapas tradicionais de processamento de consultas.

Algumas aplicações precisam fornecer informação para seus usuários de acordo com as suas localizações. Por essa razão essas aplicações consultam modelos espaciais que

permitem a assimilação dos dados. Em uma mesma aplicação, vários modelos podem ser fornecidos, por exemplo, um mapa da cidade, e a planta de um prédio. A integração destes modelos é necessária. A parte central da plataforma Nexus consiste no gerenciamento desses modelos espaciais, que representam o mundo físico. As Figuras 14 e 15 mostram as principais funcionalidades do Nexus:

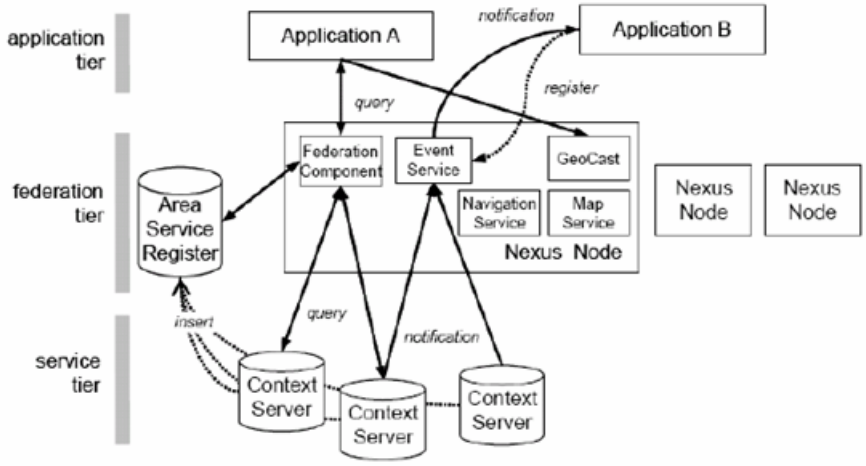

Figura 14: Arquitetura da plataforma Nexus (extraído de [BAUER, 2004]).

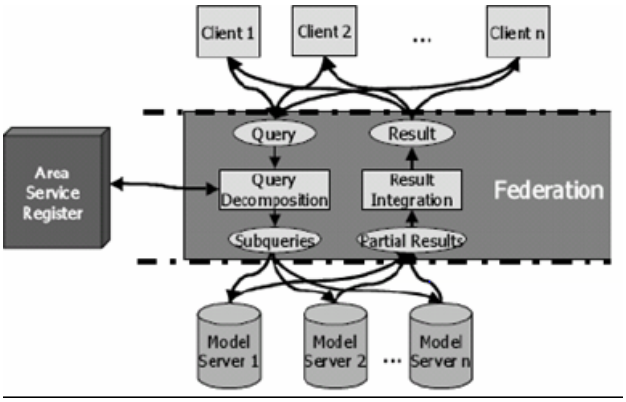

Figura 15: Arquitetura do Nexus (extraído de [BAUER, 2004]).

A plataforma Nexus consiste em três camadas (Figura 15):

1) Aplicação: são aplicações sensíveis a contexto que utilizam a plataforma para obter dados espaciais. A aplicação envia uma consulta para a federação, que a divide e envia as subconsultas para as fontes. É garantida assim transparência ao usuário, que não precisa estar ciente do ambiente distribuído. A aplicação também pode especificar para o *Event Service* receber uma notificação quando certo estado do mundo ocorrer, por exemplo, quando um usuário entrar em um edifício;

2) Federação: faz a mediação entre as aplicações e as fontes. É responsável por obter as consultas e repassá-las às fontes, e também por monitorar eventos para os quais o usuário especifica ações a serem executadas. O componente *GeoCast* armazena endereços de certas áreas geográficas, para localizar e enviar mensagens a usuários que estão nesta região;

3) Fontes (serviços): cada fonte possui sua representação de dados, que pode ser apropriada a dados estáticos, ou focadas em dados com alta taxa de atualização. Um importante ponto do projeto Nexus é fornecer uma plataforma com linguagem própria. Para modelar e acessar as informações, foram desenvolvidas duas linguagens baseadas no XML, a **AWML** (*Augmented World Modeling Language*), para modelagem do mundo real, e a **AWQL** (*Augmented World Query Language*), para consultas, mostrada na Figura 16.

```
<awql>
   <scope>
      <ecs name="nexus://nexusschemas.org/
      ContextCube" is="CC"/>
   </scope>
   <restriction>
      <equal>
         <attr name="type"/>
         <nexusdata>CC:temperatureSensor
         </nexusdata>
      </equal>
   </restriction>
   <filter>
      <includes>
         <attr name="value"/>
         <attr name="accuracy"/>
         <attr name="type"/>
      </includes>
      <excludeallother/>
   </filter>
</awql>
```

Figura 16: Consulta AWQL (extraído de [BAUER, 2004]).

Em relação às desvantagens da arquitetura, não existe uma ontologia de conceitos gerais e relações entre as várias informações contextuais, que foi criado para lidar apenas com o contexto espacial em detrimento do contexto de usuário e outros. Os dados interpretados e convertidos para alto nível são restritos às camadas de aplicação, não sendo armazenados ou reutilizáveis. O Nexus também não lida com valores históricos, com a camada federação mantendo temporariamente estas informações em memória volátil. Apesar da complexidade das linguagens criadas no Nexus, não se visualizou possíveis mudanças posteriores.

[TOENNIS, 2003] aponta o parcial poder de reconfiguração na plataforma, sendo obrigatória a extensão da hierarquia de classes padrão do Nexus. Também é necessária a

reescrita de todo o sistema caso seja necessária futura reestruturação da comunicação entre seus subsistemas [FISCHER, 2004].

4.4 Mogatu

O MoGATU foi idealizado pelo eBiquity Group, da Universidade de Maryland [PERICH *et. al.*, 2004], que tem a computação sensível ao contexto como uma de suas áreas de atuação. Esse projeto é um esforço para se implementar um *framework* para lidar com dados em ambientes inteligentes. Na visão da computação ubíqua, os sistemas automatizados irão gradativamente se integrar à vida de todas as pessoas, oferecendo a elas serviços e informações a qualquer hora e a qualquer lugar. O MoGATU está ligado a outros projetos do mesmo grupo, como o SOUPA (*Standard Ontology for Ubiquitous and Pervasive Applications*) [CHEN *et. al.*, 2004], uma ontologia para sistemas sensíveis ao contexto, e o CoBrA *(Context-aware Broker Arquitecture*) [CHEN, 2004], uma arquitetura para ambientes inteligentes baseados em agentes.

Esse projeto tem uma estratégia *peer-to-peer*, assumindo que não há um controle centralizado. O protocolo implementado no MoGATU é o CQP (*Collaborative Query Processing*), que faz com que dispositivos do ambiente ajudem outros aparelhos na localização e obtenção de respostas para consultas de uma ou mais fontes de dados. O protocolo permite que dispositivos móveis enxerguem outros dispositivos como fontes adicionais de informações. Pode-se ver tal característica por sua arquitetura na Figura 17:

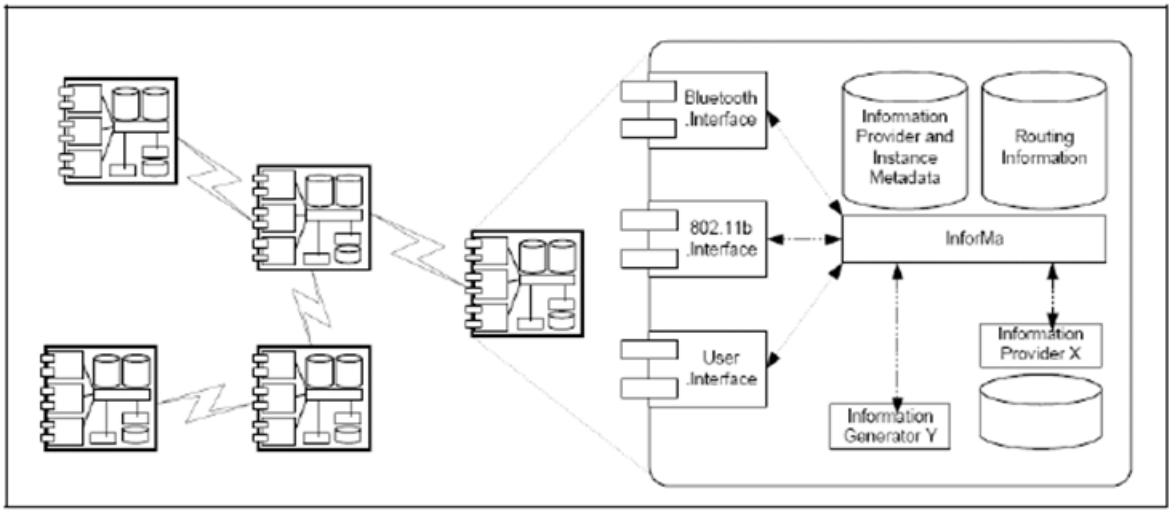

Figura 17: Arquitetura do MoGATU (extraído de [CHEN *et. al.*, 2004]).

O MoGATU trata as fontes de dados como fontes autônomas, móveis, heterogêneas e altamente distribuídas. Para resolver o problema da heterogeneidade são criadas abstrações de cada dispositivo móvel, chamadas InforMa. Estes gerenciadores são os responsáveis pela maior parte do trabalho relativo ao gerenciamento de dados e dão suporte na rede de comunicação. Cada dispositivo, seja ele um *notebook*, PDA ou sensor, deve ter o seu gerenciador. Eles são responsáveis pelas seguintes funções:

- Dar suporte às consultas realizadas pelos consumidores de informação, inclusive verificando se as informações estão atualizadas;
- Manter *cache* de consultas passadas, caso os dispositivos estejam indisponíveis. Entretanto, nem todos os dispositivos terão este recurso, por limitações de performance. Segundo [PERICH *et. al.*, 2005], pode ser impossível manter todos os dados consistentes, o que é uma desvantagem em certos domínios;
- Incluir um perfil de usuário refletindo suas preferências e necessidades, e usa esse perfil para adaptar suas estratégias armazenadas e iniciar uma colaboração com vizinhos para obter a informação desejada;
- Descobrir a localização e disponibilidade de outros dispositivos na vizinhança.

Segundo os autores, os nós se comunicam com linguagens enriquecidas através de ontologias, o que resolve o problema da falta de um esquema global. Os consumidores de informação são representados por entidades, sejam humanos ou sistemas, que fazem consultas e atualizam os dados do ambiente. A ontologia SOUPA, utilizada no MoGATU, é expressa usando OWL e são incluídos módulos de componentes de vocábulos para representar agentes inteligentes com algumas informações associadas, como crença, desejos, intenção, tempo, espaço, eventos, perfis de usuários, ações e políticas para segurança e privacidade. Para se construir as consultas no ambiente MoGATU, pode-se notar que, além das cláusulas convencionais SQL, existem outras cláusulas como a *TIME* para aspectos temporais.

Entre as desvantagens existentes, uma consulta é sempre encaminhada a um InforMa que caso não possua a informação desejada, a procura em seus *peers* vizinhos e assim sucessivamente, o que pode levar a um tempo elevado para a localização dos dados. Esse problema é particularmente crítico em domínios tais como a telemedicina. Não foi observado o tratamento para consultas baseadas em evento. O MoGatu, no que se refere aos requisitos de dados contextuais, não foi criado com o propósito de ser extensível, ou seja, atender novas

aplicações. Também não são providos maiores detalhes quanto ao armazenamento e uso de estatísticas sobre acesso de aos dados por cada usuário.

4.5 MoCA

No MoCA [SACRAMENTO *et. al.*, 2004] a proposta é uma arquitetura de *middleware* baseado em serviço, configurável e extensível para facilitar o desenvolvimento de aplicações sensíveis ao contexto distribuídas, onde os serviços são modulares e passíveis de distribuição, fáceis de configurar e atualizar. A arquitetura do MoCA é apresentada na Figura 18:

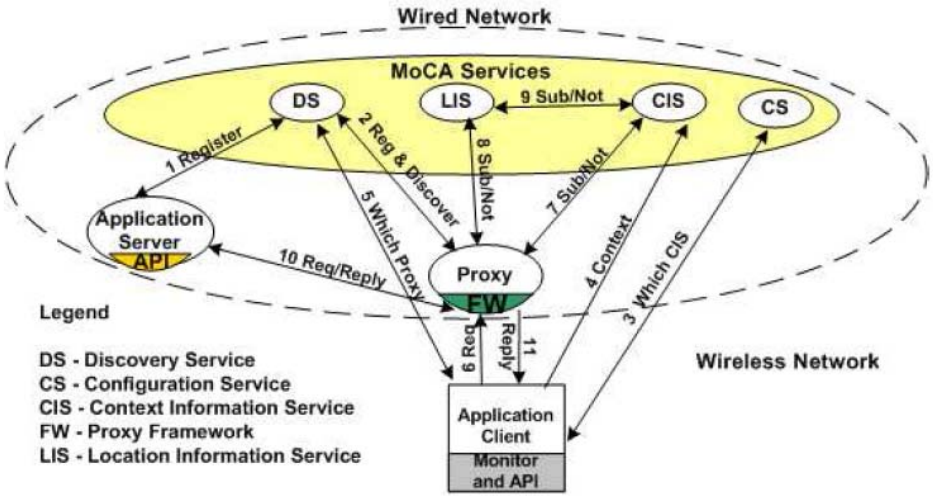

Figura 18: Arquitetura do MoCA (extraído de [SACRAMENTO *et. al.*, 2004])

A Figura 18 também apresenta a seqüência básica de interações entre os elementos da arquitetura, onde é ilustrado os papeis executados por cada elemento durante o registro e a execução de aplicações colaborativas, compostas de um ou mais instâncias de *Application Servers*, *Proxyies* e *Application Clients*.

Inicialmente o *Application Server* se registra no DS (*discovery service*) (passo1) informando o nome e as propriedades do serviço colaborativo que ele implementa. Cada Proxy da aplicação também faz um registro similar no DS (passo2). Desse modo os *Application Clients* podem consultar o DS para descobrir como acessar um dado serviço colaborativo na rede corrente. O monitor, executando em cada dispositivo móvel, observa o estado das fontes locais e o sinal RF, e envia essa informação contextual para o CIS (*Context information service*). O endereço do CIS alvo e a periodicidade para enviar as informações de

contexto são obtidos do CS (*configuration service*) quando o Monitor é inicializado (passo 3). Então, o monitor envia periodicamente o estado da informação para o CIS (passo4).

Depois de descobrir um *Proxy* que implementa o serviço colaborativo desejado direto do DS (no passo 5), o cliente pode começar a enviar requisições para o *Application Server*. Cada requisição é roteada para o *Proxy* correspondente (passo6), que processa a requisição do cliente com as adaptações necessárias para a aplicação, e encaminha para o *Application Server*. Por exemplo, o *Proxy* pode enviar um Interest Expression para o CIS (passo 7) registrando esse interesse em notificações de eventos sobre mudanças de estados do cliente que ele representa. Um *Interest Expression* pode ser, por exemplo, { *"FreeMem < 15% OR roaming=True*}.

Agora, sempre que o CIS recebe uma informação de contexto de um dispositivo, ele verifica se esse novo contexto se encaixa em algum *Interest Expression* armazenado. Se isso acontecer o CIS gera uma mensagem de notificação e envia para todos os *Proxies* que registraram esse interesse.

Aplicações que requerem informação de localização podem usar diretamente o LIS para obter essa informação ou, se quiserem informações periódicas, podem registrar esse interesse no CIS que usará o LIS (*location information service*) para inferir a localização do dispositivo e enviar a notificação correspondente para o *Application Proxy*.

Quando o *Application Server* recebe requisições do cliente, a requisição é processada e resposta é enviada para alguns ou todos os *Proxies* (passo 10), que pode modificar/processar a resposta de acordo com a notificação recebida do CIS sobre o dispositivo móvel correspondente. Por exemplo, se a conectividade estiver baixa ele pode armazenar os dados em um *buffer* para envio posterior. Além disso, o *Proxy* pode usar outras informações de contexto, como a localização, para determinar qual a data, quando e como deve enviar para o cliente no dispositivo móvel (passo 11).

O problema dessa abordagem é o grande fluxo de informações que devem ser enviadas para o CIS, ou seja, em um sistema com muitos dispositivos onde todos ficam enviando seus *status* periodicamente para o CIS pode ser tornar extremamente pesado. Além disso, essas informações que estão sendo enviadas podem não ser de interesse de ninguém e mesmo assim estão gerando tráfego de rede. Ainda assim, o MoCA compartilha o objetivo de uma arquitetura de propósito geral, e componentes modulares disponibilizados.

4.6 AWARENESS

Desenvolvido em parceria entre diversas instituições holandesas, dentre elas o CTIT (*Centre for Telematics and Information Technology*) e o Telematica Instituut da University of Twente, o AWARENESS (*context AWARE mobile Networks and ServiceS*) [WEGDAM, 2005] tem como principal objetivo projetar uma infra-estrutura de suporte a serviços e aplicações sensíveis ao contexto. Como domínios de aplicação utilizados para validar a infra-estrutura desenvolvida, são utilizados cenários e aplicações médicas reais. O AWARENESS busca integrar serviços e dispositivos da computação ubíqua com o uso de técnicas e mecanismos de processamento de informações de contexto e suporte à pró-atividade das aplicações, além de ontologias em metodologias de descoberta de serviços. A infra-estrutura ainda em desenvolvimento deverá prover suporte à mobilidade em ambientes sensíveis ao contexto, além de novos métodos de inferência e uso de contexto em domínios variados [PESSOA, 2006]. Seu módulo de gerenciamento de dados é visto na Figura 19.

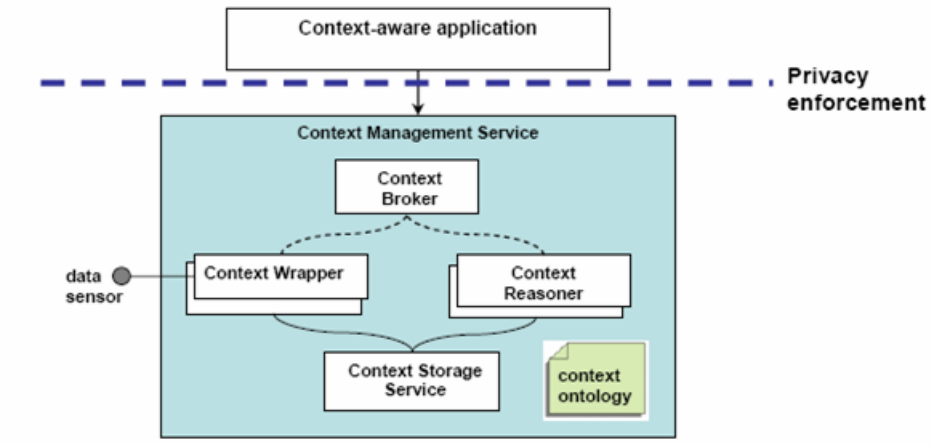

Figura 19: Gerenciamento de dados no Awareness (extraído de [COSTA, 2003])

Os *Context Wrapper*, *Context Storage Service*, e *Context Reasoner* representam fontes de dados contextuais e são implementados com a mesma interface de acesso. O gerenciador sabe quais tipos de contexto estão sendo manipulados por cada um através de uma ontologia. O *Context Broker* é responsável por realizar a descoberta das fontes.

Para a integração de dados, o Awareness também possui entidades responsáveis pela interface superior (com as aplicações) e inferior (com as fontes). O Awareness possibilita a entrega ativa de dados, obedecendo ao padrão de projeto ECA. Este padrão desacopla os três estágios de regra vistos anteriormente, o que fornece grande flexibilidade. As ações

executadas podem ser (i) chamadas a *web Services*; (ii) resposta para a aplicação ou (iii) execução de serviço interno. A estrutura responsável por sua execução é chamada *ECA Controlling Service*, inspirado em *middleware* existentes com a arquitetura *publish-subscribe* [SINDEREN *et. al.*, 2006]. Em relação a operadores, o Awareness possui expressividade para eventos compostos através de lógica booleana, além de aceitar predicados sob demanda.

O projeto Awareness utiliza padrões de projetos tais como o *Chain of Responsability* e o *Observer* para a modelagem dos monitores de eventos. Esta plataforma possui extensibilidade para acréscimo de operadores e representação de eventos compostos. Decidiu-se por um módulo para monitoramento, ao invés de uma máquina de execução de consultas.

O sistema mantém um sistema de histórico, além de um *log* que, combinado com um algoritmo de *data mining*, descobre padrões sobre as consultas pelos usuários: quais são as consultas mais realizadas, e em quais horários são feitas. As interfaces fornecidas são somente para adição de tipo de fonte, registro e descoberta de fontes.

Uma tabela comparativa referente aos requisitos atendidos pelas arquiteturas citadas é fornecida, onde se pretende demonstrar o diferencial da arquitetura descrita neste trabalho.

Tabela 18: Comparação de requisitos para plataformas sensíveis ao contexto.

	Awareness	**Nexus**	**MoCA**	**DBGlobe**	**Mogatu**
Extensibilidade	**X**	**X**	**X**		**X**
Heterogeneidade	**X**		**X**	**X**	**X**
Distribuição	**X**		**X**		**X**
Metadados	**X**	**X**	**X**	**X**	**X**
Dinamismo das fontes		**X**	**X**	**X**	**X**
Regras ACID		**N.E.**			
Perfil de usuário	**X**	**N.E**	**X**	**X**	**X**
Entrega ativa de dados	**X**	**X**	**X**	**X**	**X**
Linguagem de eventos	**X**	**X**	**X**	**X**	**X**
Eventos compostos	**X**				
Distribuição de evento					
Criação de operadores					
Prioridade entre cons.		**N.E.**			
Histórico	**X**	**N.E.**			
Inicialização dinâmica de fontes	**N.E.**	**X**	**X**	**N.E.**	**X**
Mobilidade		**X**	**X**	**N.E.**	**X**
espaço-temporal		**X**		**X**	**X**
Linguagem eventos	**X**	**X**	**X**	**X**	**X**
Linguagem espacial		**X**			

Do que foi observado nos trabalhos anteriormente citados, todos possuem um conjunto de metadados reduzidos. Muitos foram projetados para domínios específicos, o que pode

dificultar seu uso em diferentes domínios. A arquitetura proposta neste trabalho se diferencia dessas ao permitir a configuração e extensibilidade de seus componentes. A modularidade e reuso facilitam o seu uso em novas aplicações decorrentes da ampla variedade de tipos de contexto existentes, além de permitir a incorporação de novos requisitos.

4.7 Outros Trabalhos Relacionados

Diferentemente dos trabalhos apresentados anteriormente, foram estudados vários outros projetos que se concentram em determinados requisitos da computação sensível ao contexto apresentados no capítulo 3. Tais projetos constituem estudos valiosos sobre diversas abordagens para a solução dos requisitos apontados no capítulo anterior.

a) Liquid [HEES *et. al.*, 2003]: das abordagens encontradas, é o que mais utiliza este conceito de processamento de consulta. Em sua visão, um serviço de consultas distribuído, com dados e processamento distribuídos, é o próximo passo natural para a infra-estrutura de contexto a fim de melhorar a escalabilidade. Um dos benefícios da incorporação de processamento de consultas na computação sensível ao contexto é melhor eficiência da infra-estrutura, além de se evitar re-trabalho e trocar informações e desafios entre as duas comunidades [HEER *et. al.*, 2003]. Entretanto, perde em flexibilidade em relação ao framework QEEF utilizado nesta dissertação como pode ser visto na tabela 19. No caso, o CoDIMS possibilita modos de execução e linguagem de retorno dos resultados variável.

Tabela 19: Comparação entre abordagens de detecção de eventos

	PEC	***consultas distribuídas***	***Linguagem de consulta***	***Linguagem de resultado***	***Comunicação***	***Mapear tipos***	***Operador***	***Modo execução***
Liquid	Sim	Sim	XML	XML	HTTP	Path semi-estruturado	Escalavel	Fixo
CoDIMS	Sim	Sim	Variável	Variável	Web Services	Metadados	Variável	Variável

b) Cougar [BONNET *et. al.*, 2000]: utiliza conceito de relações virtuais para incluir sensores em planos de execução de consulta, sendo utilizado normalmente, e ligado por operadores a fontes tradicionais na árvore de execução de consulta. Seu modelo de custo inclui energia disponível nas fontes e comunicação entre nós, além dos encontrados em [OZSU; VALDIUREZ, 2001]. O projeto Cougar foi criado devido ao rápido aumento no uso e

disponibilidade de sensores. Seu foco são redes de sensores que combinam diversos dados, como temperaturas e dados sísmicos, através da criação de uma camada de gerenciamento de dados distribuídos com algoritmos apropriados à computação com limitações de energia e largura de banda.

c) [PAYTON *et. al.*, 2005]: considera contexto como sendo um repositório que reflete as constantes mudanças do ambiente da aplicação, e cujo programador deve consultar a fim de ganhar acesso às informações. Seu estudo de caso é na automação aplicada ao agro negócio. Sua plataforma lida com validade temporal de dados, além de considerar deleção de tuplas.

d) M3: é uma arquitetura *context-aware* tem por meta suportar interoperabilidade com outros componentes, tais como aplicações legadas e novos componentes. Novos componentes devem poder exibir novos tipos de interação além do RPC tradicional (*remote procedure call*). Tal funcionalidade é realizada através de um Coordenador e um Gerente de Adaptação. [RAKOTONIRAINY *et. al.*, 2001].

e) ActiveCampus [GRISWOLD *et. al.*, 2003]: reconhece que serviços, fontes, tipos de contexto, formatos de dado e dispositivos de acesso devem ter interfaces padrão. Entretanto, ele não menciona novas funcionalidades para a arquitetura.

f) HiPAC [DAYAL *et. al.*, 2004]: idealizado pela IBM, divide os estágios de condição e ação, possibilitando que executem em transações distintas, em diferentes momentos e com tratamentos de exceção apropriados. Existe, naturalmente, um gerenciador de eventos. A parte de ação da regra, por exemplo, pode tentar ser executada novamente, disparar uma ação alternativa ou mesmo abortar o processo [PATON; DIAZ, 1999].

g) Hermes [VARGAS *et. al.*, 2004] [PIETZUCH, 2004]: *possui Event Clients*, que disponibilizam e requisitam notificações, para gerenciamento das notificações, e event broker, que fazem a propagação das notificações. As notificações são enviadas via mensagens XML. O Hermes possui adaptadores com a função de converter o evento ocorrido no SGBD para este formato XML. Ele possui uma arquitetura distribuída que segue o paradigma *publish-subscrube*, frequentemente utilizado nas plataformas sensíveis ao contexto. O Hermes possui a desvantagem onde eventos compostos só podem ser identificados em um mesmo nó.

h) Sentinel [CHAKRAVARTHY *et. al.*, 1994]: é um SGBD orientado a objetos, que provê a detecção de eventos através de uma linguagem expressiva e regras ECA. Foi o primeiro projeto a criar uma linguagem para eventos compostos.

i) Starburst [WIDOW, 1996]: projeto para detecção de eventos, em como meta a escalabilidade. Possui sofisticados mecanismos para controle de concorrência, recuperação em erros, e definição de prioridade entre regras.

J) [JAEGER; OBAMAIER, 1997]: fornece uma sintaxe para a detecção de eventos compostos, com a intenção de se obter os diversos tipos de paralelismo do Capítulo 2.

k) SAMOS [GATZIU; DITTRICH, 1994]: utiliza Redes de Petri para a definição de eventos, onde cada estado corresponde a uma ocorrência de um tipo de evento.

l) Snoop [BERNAUER *et. al*., 2004]: é um projeto mais recente, onde documentos XML são utilizados para a representação de eventos distribuídos, com a extensibilidade e expressividade que esta linguagem oferece.

m) CREAM [CILIA *et. al.*, 2003]: utiliza ontologias para representação dos eventos provindos de vários sistemas.

n) [MOTAKIS AND ZANIOLO, 1995]: mostra um repositório de ocorrências similar ao proposto nesta dissertação, e mostrado na Figura 20. É válido ressaltar que em bancos de dados ativos, uma ocorrência de evento significa uma transação, alterações em tuplas. Na computação sensível ao contexto, são enxergados como um acontecimento de interesse do usuário.

hist_monit	EventType	TableName	TimeStamp	Stage
	nil	nil	0000	0
	upd	ACC	1423	1
	upd	ACC	1425	2
	ins	ACC	1430	3
	ins	ACC	1502	4

Figura 20: Repositório de eventos (extraído de [MOTAKIS, 1995]).

É possível comparar na Tabela 20 os aspectos atendidos pelos projetos descritos acima. Todos os projetos analisados possuem uma linguagem de representação de eventos, mas nem todos permitem a criação de novos operadores.

Tabela 20: Repositório de eventos (extraído de [MOTAKIS, 1995]).

	Hipac	***Snoop***	***Liquid***	***CoDIM-CA***
Linguagem específica	Sim	Sim	Sim	Sim
Eventos compostos	Não	Sim	Sim	Sim
Distribuição dos	Não	Não	Sim	Sim

	Hipac	***Snoop***	***Liquid***	***CoDIM-CA***
operadores entre nós				
Novos operadores	Não	Sim	Não	Sim

o) [SHIJUN *et. al.*, 2005] e [IFTIKHAR *et. al.*, 2006]: apresentam estratégias de processamento de consultas para sistemas sensíveis ao contexto. Estes trabalhos sugerem que a consulta exige o estágio de reescrita, com uso de dados de sensores, perfis e ontologias de domínio. Entretanto, estes projetos apenas gerenciam subcrições *request-response*.

Recentemente têm-se intensificado os estudos metadados de uma maneira geral, em áreas desde a Engenharia de Software até a Inteligência Artificial. Sua relevância vem sendo consolidada, por exemplo, através da criação de linguagens para tal finalidade.

p) OBSERVER [MENA *et. al.*, 2000]: é um sistema para processamento de consultas desenvolvido na Universidade Politécnica de Madrid, Espanha. OBSERVER tem como objetivo principal o processamento de consultas em sistemas de informação globais, baseado em interoperabilidade entre ontologias pré-existentes. Ele utiliza metadados para capturar o conteúdo das informações dos repositórios. Falta a ele, entretanto, a mesma abordagem para fontes de dados contextuais, móveis e altamente dinãmicas.

q) SOUPA [CHEN, 2004]: ontologia criada pelo grupo de pesquisadores eBiquity através da linguagem OWL para auxiliar aplicações ubíquas. Ela é utilizada pelo projeto MoGatu na representação de suas fontes. Esta ontologia representa características como dispositivos computacionais limitados, sua localização e eventos relacionados. Entretanto, é focada em agentes e representações de crenças e intensões (BDI), omitindo algumas informações sobre fontes de dados e autoria das fontes de dados.

r) [GRAY; SALBER, 2001]: utiliza informações sobre a qualidade de sensores, como frequência e precisão, mas não com o intuito de integração de dados.

s) [HÖNLE *et. al.*, 2005]: aponta as questões de autoria sobre as fontes de dados como necessárias e como utilizá-las na plataforma Nexus.

t) Exodus [GRAEFE; DEWITT, 1987]: utiliza prioridades de consultas em seu otimizador de consultas.

u) MOEM [GERGATSOULIS; STRAVAKAS, 2003]: apresenta a linguagem de consulta MOEM (*multidimensional OEM*), com expressividade suficiente para representar o histórico de mudanças em um documento XML, além da história do próprio *Schema* XML.

Possui também uma sintaxe para a representação de dados e realização de consultas de acordo com perfis. Um exemplo desta sintaxe é vista a seguir na Figura 21, onde diversos códigos são atribuídos a um livro, de acordo com seu país de publicação:

```
<book><@isbn>
          [edition=greek]<isbn>0-13-110370-9</isbn>[/]
          [edition=english]<isbn>0-13-110362-8</isbn>[/]
</isbn></book>
```

Figura 21: Representação de perfil no projeto MOEM (extraído de [GERGATSOULIS; STRAVAKAS, 2003]).

v) [STEFANIDIS *et. al.*, 2005][PITOURA; STEFANIDIS, 2004]: reconhece a necessidade de se integrar informações de perfil e contexto do usuário em bases de dados relacionais. Contextos são modelados como parâmetros, normalmente de um conjunto discreto, e que tem um estado (valor) em um dado momento. Pode-se imaginar um cubo para a visualização destes dados. Existe uma divisão entre preferências simples, que são as que dependem de apenas uma informação contextual {restaurante=grego} e compostas, que dependem de uma combinação linear de várias preferências {0.6 restaurante=grego + 0.7 dia=sol}. Estes trabalhos apontam que as informações de perfil de usuário podem até mesmo ser utilizadas na otimização de consultas, possibilitando a entrega mais veloz dos resultados mais relevantes.

x) [KOUTRIKA; IOANNIDIS, 2004]: lida com o conceito de preferência positivo (aprovação) e negativo (reprovação), além da prioridade entre as diversas informações de perfil. Alguns tipos de dados de perfil existentes são dados de contas de usuário para os sistemas, informativos, interesses. Perfis de grupos também podem ser criados, simplificando a atribuição de acessos e preferências a vários usuários com algum relacionamento.

w) [GERGATSOULIS; STRAVAKAS, 2003][NATH, 2002]: utiliza o conceito de janela de armazenamento, ou seja, um *buffer* de informações históricas é mantido, fora do qual elas são descartadas. Estas considerações serão utilizadas no desenvolvimento de um módulo para histórico contextual.

y) [MANTORO; JOHNSON, 2003]: define um modelo de baixo custo para o armazenamento de histórico de localização de usuários.

z) [MAYRHOFER, 2004]: aponta que dois níveis de histórico podem ser armazenados, um local, que não necessita de um controle central, e outro para operações de detecção de padrões e predição de valores futuros, cuja escalabilidade e necessária.

4.6 Conclusão

A área de aplicações móveis sensíveis ao contexto ainda não foi devidamente explorada pela comunidade de banco de dados. Especificamente sobre sistemas de integração de dados contextuais, são encontrados poucos projetos na literatura, que não atendem a todos os requisitos para uma plataforma flexível como o INFRAWARE.

Para solucionar este obstáculo, extensibilidade e configuração foram obtidas a partir do uso de tecnologias apropriadas como *web services* e XML e metodologias de desenvolvimento baseado em componentes. Estas características auxiliam significativamente no desenvolvimento dos demais módulos, vistos no Capítulo 5.

5 Análise e projeto da arquitetura

Com base nos requisitos listados no capítulo 4, foi realizado o projeto e análise da arquitetura, para que atenda as necessidades de sistemas sensíveis ao contexto.

Neste capítulo será apresentada uma arquitetura para integração de dados contextuais, além de toda a análise e projeto, através de diagramas de casos de uso, componentes, classes e seqüência apresentados na linguagem UML. Serão focados a extensiblidade, coesão e baixo acoplamento entre os componentes da arquitetura.

5.1 Diagramas de Casos de uso

Cada requisito funcional mostrado anteriormente dá origem a casos de uso que são mostrados nesta seção. Os requisitos não-funcionais, por sua própria definição, não dão origem a casos de uso, mas são presentes em toda a arquitetura. Os casos de uso identificados para os requisitos levantados são:

- **Configuração e escalabilidade**: a arquitetura deve permitir modificar ou acrescentar novos componentes, personalizados para cada aplicação. Existem dois tipos de configuração: *i*) Configuração física, que determina quais componentes estão presentes em cada aplicação; *ii*) Configuração lógica, define um workflow de operações dos componentes existentes que devem ser executadas para cada uma das tarefas (por exemplo, execução de uma subscrição *request-response*) permitidas. O ator envolvido nestes casos de uso é o projetista que define/configura o ambiente para uma dada aplicação (Figura 22).

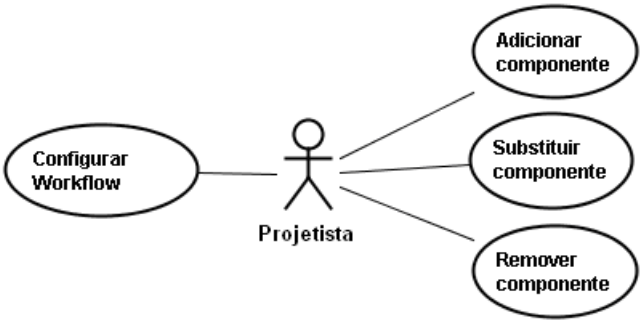

Figura 22: Casos de uso: configuração

- **Metadados:** estando o ambiente configurado, para cada fonte de dados adicionada, seus metadados e estatísticas também devem ser registrados. Metadados devem ser removidos quando uma fonte de dados se tornar indisponível, para que o conjunto de metadados se mantenha consistente e atualizado. E estatísticas devem ser permanentemente mantidas atualizadas. O ator responsável é o administrador do ambiente, tal como um DBA (Administrador de Banco de Dados).

Figura 23: Casos de uso: metadados

Por exemplo, uma informação importante que se reflete na instanciação correta do agente responsável é o envio automático ou não de dados. Alguns sensores possuem capacidade de enviar seus próprios eventos, como os *Holter* utilizados pelo projeto TeleCardio. Um sensor de localização utilizado em uma aplicação de turismo, por sua vez, pode não possuir esta capacidade. Outra propriedade importante é o tipo do atributo: normal, estático, sensor.

- **Entrega ativa de dados:** os modos de entrega de dados permitidos para uma subscrição são: *request-response*, *time-driven*, *event-driven* e *real-time*. A detecção de um evento só será terminada, ou seja, um agente só será destruído quando a subscrição for cancelada pelo usuário, ou caso o campo *timeout*, presente na subscrição de controle, seja atingido. O ator responsável por submeter uma subscrição é o usuário final do sistema.

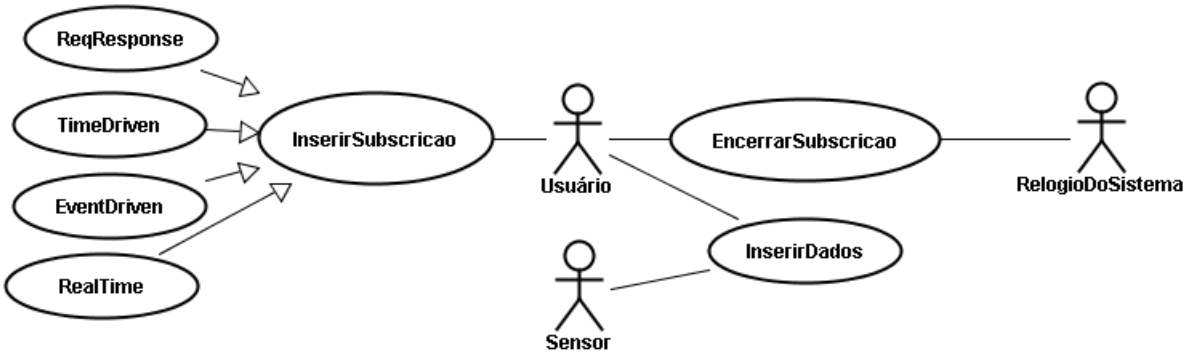

Figura 24: Casos de uso: inserção de subscrições

- **Perfil de usuário:** referem-se às modificações dos perfis que representam os usuários. Estes perfis possuem importante papel na otimização de consultas. O formato do arquivo de perfil deve permitir a tratabilidade, ou seja, recuperar o motivo de uma inferência na otimização de consultas.

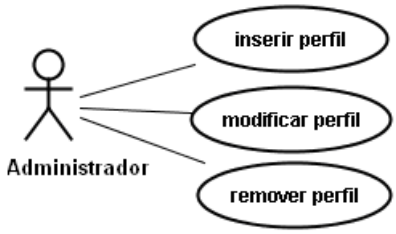

Figura 25: Casos de uso: perfil de usuário

- **Histórico contextual:** armazenam-se dados contextuais passados ou eventos de interesse corridos. Em relação aos eventos que devem ser monitorados no cenário deste trabalho, exemplos são os sinais vitais, ou a chegada de um médico a um hospital.

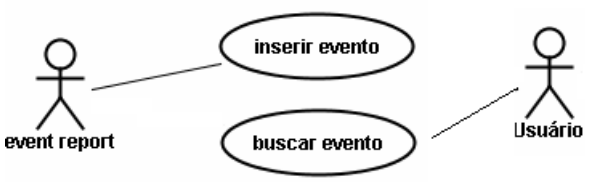

Figura 26: Casos de uso: armazenamento de eventos

Em relação aos dados históricos, o usuário determina a configuração do armazenamento de histórico, seja a freqüência de gravação, tamanho da janela de *buffer*, ou o momento no qual o armazenamento se iniciou.

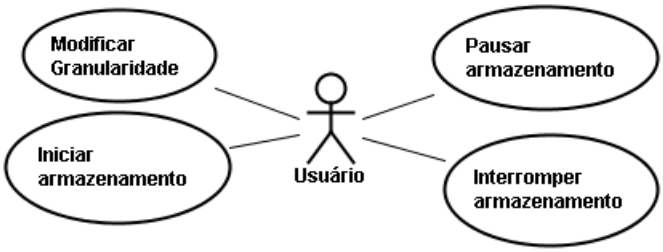

Figura 27: Casos de uso: armazenamento de histórico

- **Prioridade entre consultas:** Somente o administrador do sistema deve ter acesso às prioridades de consulta. O Gerente de Tarefas é responsável por checar estas prioridades e priorizá-las dinamicamente.

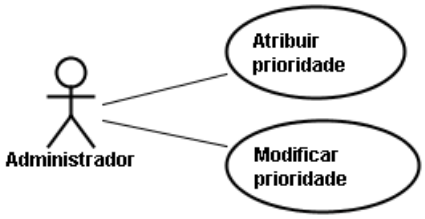

Figura 28: Casos de uso: priorização entre consultas

- **Linguagem de eventos:** A arquitetura deve prover facilidade de adicionar um novo operador de consulta através da inserção de seu código e atualização do DTD utilizado pela MEC para representação de metaplanos e listagem de operadores disponíveis.

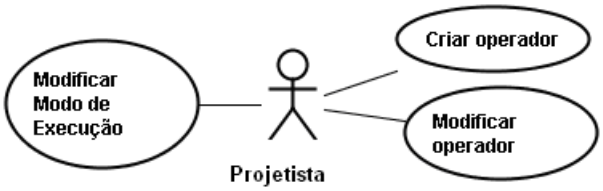

Figura 29: Casos de uso: operadores

- **Gerência de fontes:** responsável pelo monitoramento dos *wrappers*, fontes de dados e seus *status*, podendo ser consultados com uma linguagem em alto nível. É utilizado para gerenciamento do ambiente.

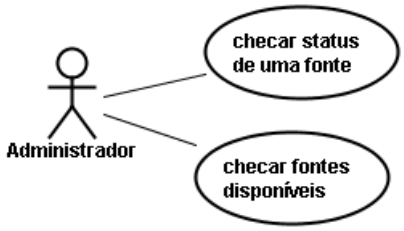

Figura 30: Casos de uso: gerência de fontes

5.2 Visão conceitual

Após o levantamento de requisitos para integração de dados contextuais e dos casos de uso vistos anteriormente, percebeu-se que o CoDIMS, em sua versão original, atende a alguns destes requisitos. Neste sentido, uma nova instanciação do CODIMS foi proposta para realizar a função de acesso e integração de dados na plataforma Infraware, aqui denominado de CoDIMS-CA. A reutilização e adaptação de seus componentes para diferentes domínios de aplicação constituem os pontos fortes para o seu uso em plataformas de propósito geral, e em particular as para computação sensível ao contexto.

O CoDIMS-CA, seus componentes definidos e interações entre eles é apresentado na Figura 31. Os principais componentes originais do CoDIMS, conforme definido em [BARBOSA, 2001] são: *i*) Controle, que gerencia a configuração e execução do sistema; *ii*) Metadados, que contêm todas as informações relativas às fontes de dados; *iii*) Acesso a Dados, que faz o acesso a cada fonte de dados a partir das informações descritas nos metadados; *iv*) Processamento de Consultas, responsável pela análise, reescrita, otimização e execução das consultas.

Originalmente o CoDIMS apresentava apenas consultas do tipo *request-response*. Para atender aos requisitos das aplicações sensíveis ao contexto, tornou-se necessária a incorporação de novos módulos componentes, que são: *v*) Gerente de tarefas, que prioriza e gerencia a execução das requisições provenientes das aplicações; *vi*) Agentes, que monitoram as fontes a fim de prover comportamento ativo ao sistema integrador de dados; *vii*) Containers, que auxiliam na execução distribuída da MEC, gerenciando os operadores e agentes presentes em cada dispositivo; *viii*) Histórico contextual, que realiza o armazenamento e consulta dos dados extraídos das fontes; *ix*) *Event Report*, registro e

consulta de eventos detectados pelos agentes; *x*) Perfis, para gerenciamento de perfis individuais para uma entrega de resultados adequada aos anseios do usuário.

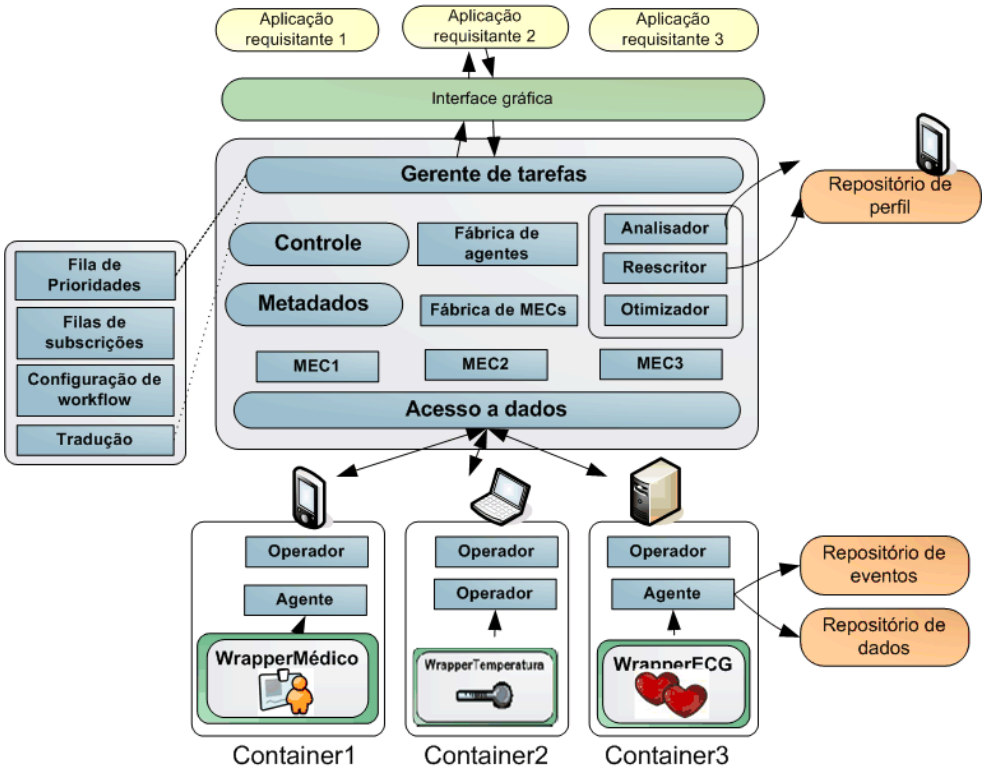

Figura 31: Arquitetura Proposta para consultas a dados contextuais

O fluxo de dados será mostrado sucintamente na seção 5.7. Além disto, o capítulo 6 contempla um estudo de caso e todo o fluxo durante sua execução.

5.3 Diagrama de Componentes

Um detalhamento maior da visão conceitual mostrada anteriormente é o seu diagrama de componentes correspondente. O diagrama de componentes na Figura 32 fornece uma visão mais aprofundada do sistema. Entre eles se encontram os pacotes já existentes no CoDIMS onde não foram realizadas modificações, e os novos módulos, como a MEC distribuída e os agentes. Este conjunto de funcionalidades é um ponto forte em relação às outras arquiteturas, como visto nos estudos comparativos realizados, e se considerando a ampla variedade de domínios que necessitam ser atendidos.

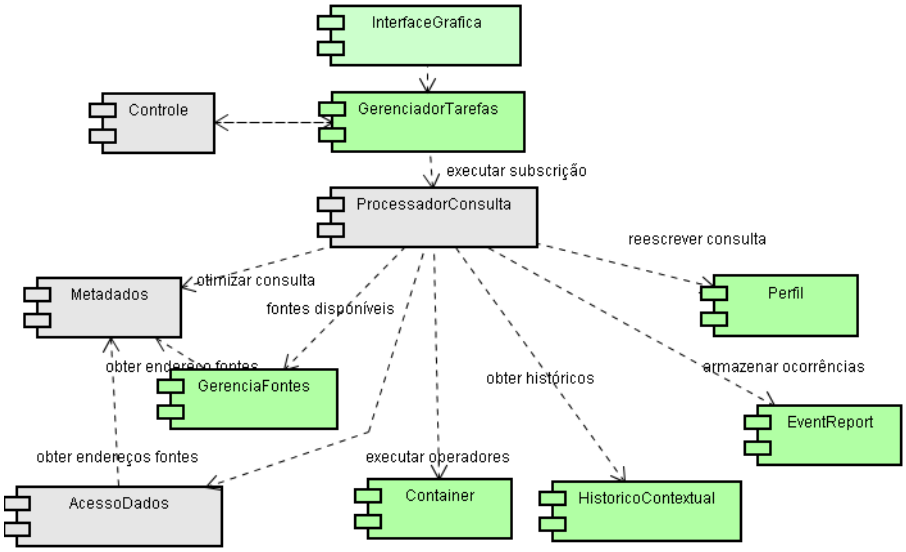

Figura 32: Diagrama de componentes

Na Figura 32 pode-se visualizar a dependência de grande parte dos componentes em relação ao processador de consulta.

5.4 Diagrama de classes

Nas próximas seções, serão descritas individualmente todas as principais classes modeladas e implementadas. Na figura 33 é apresentada uma visão geral do diagrama de classes.

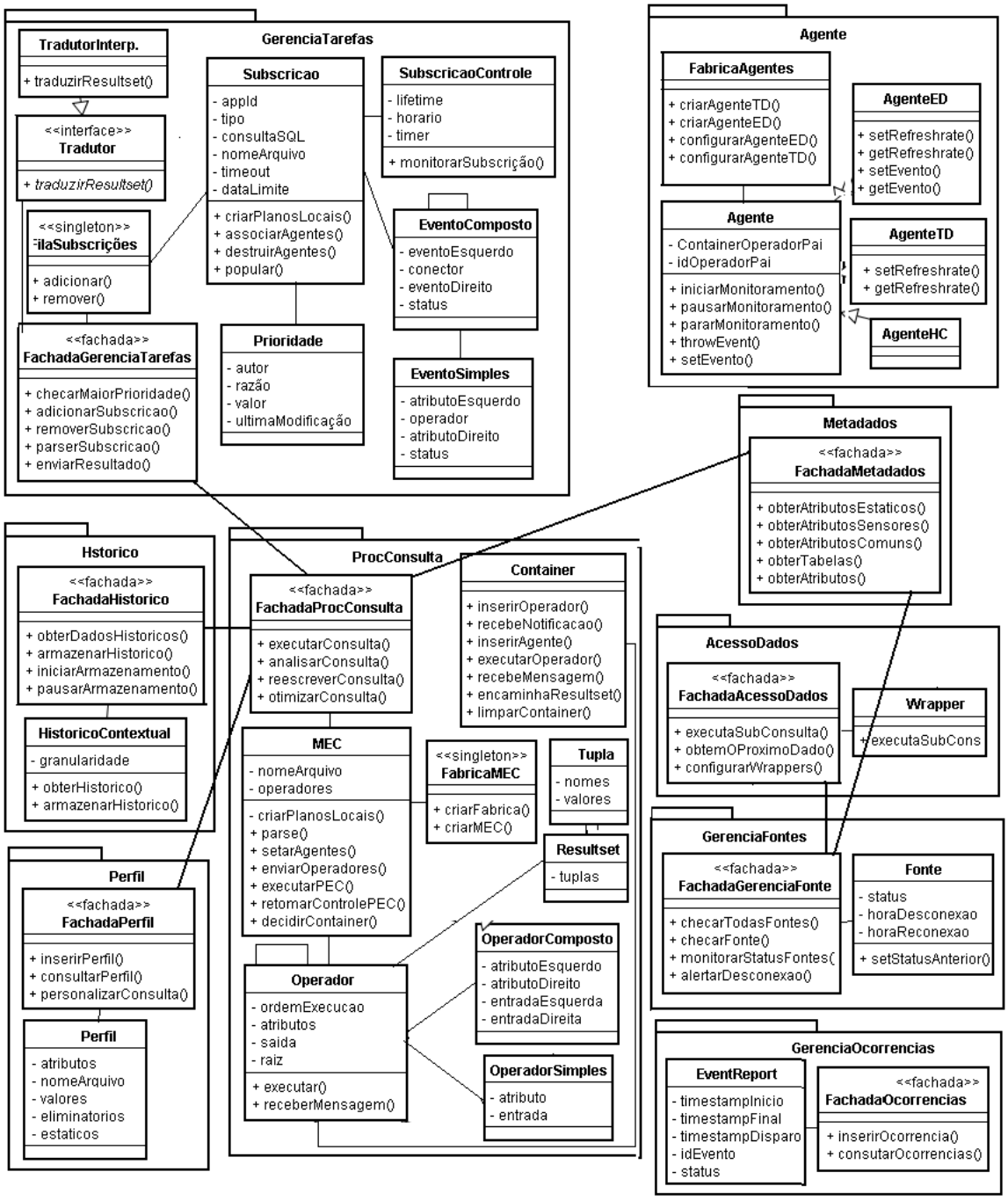

Figura 33: Diagrama de classes

A capacidade de adaptação da arquitetura aqui proposta pode ser vista em todos os estágios do processo de integração de dados: na otimização de consultas, para atender às prioridades de cada cliente, em um ambiente onde planos de consulta estáticos são inviáveis; em linguagens de consulta, de retorno de resultados e eventos, de modo a

fornecer sob demanda sintaxes convenientes a cada domínio; em metadados e estatísticas. A figura 34 mostra como diversas estratégias podem ser utilizadas em cada módulo da arquitetura. Por exemplo, novos metadados podem ser suportados com alterações da ontologia que definem os metadados.

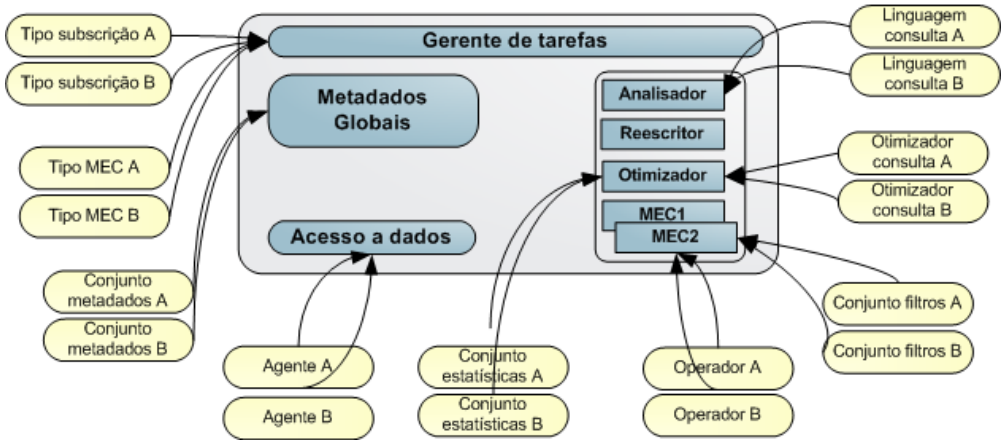

Figura 34. Pontos de configuração da arquitetura

5.4.1 Pacote *Wrapper*

Um *wrapper*, também conhecido como *Adapter* [GAMMA *et. al.*, 1995] é um *software* envoltório, para acoplamento de um componente previamente desenvolvido a um sistema externo. Ele constrói a ligação entre o sistema externo e o sistema alvo através da tradução de comandos e dados trocados entre ambos os sistemas [GAMMA *et. al.*, 1995]. As informações armazenadas sobre os *wrappers* são, dentre outras: linguagem de programação utilizada, seus métodos (com parâmetros de entrada e saída), e quantidade de memória utilizada pelo *wrapper* para realizar sua função.

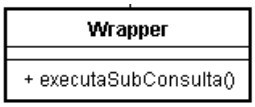

Figura 35. Pacote *Wrapper*

5.4.2 Pacote Fábrica de Agente

A classe Fábrica de Agentes (Figura 36) segue a mesma lógica da Fábrica de *Wrappers*, possuindo métodos para a criação e configuração dos três tipos de agentes. A

configuração ocorre no momento após a análise da subscrição, a fim de enviá-los aos *containers*.

Estendendo a classe *Thread*, a superclasse Agente controla principalmente seus eventos associados e o próprio fluxo de execução. Possui também uma associação com seu operador relacionado, para encaminhar o resultado quando da ocorrência de um evento. O padrão de projeto *Observer* [GAMMA, 1995] define uma dependência um-para-muitos entre objetos, de maneira que quando um objeto muda de estado todos os seus dependentes são notificados e atualizados. Esta é uma estrutura apropriada para a árvore de detecção de eventos e comunicação entre os agentes. O principal método é:

- throwEvent(): ocorre quando o agente captura um evento e repassa esta notificação para o evento pai.

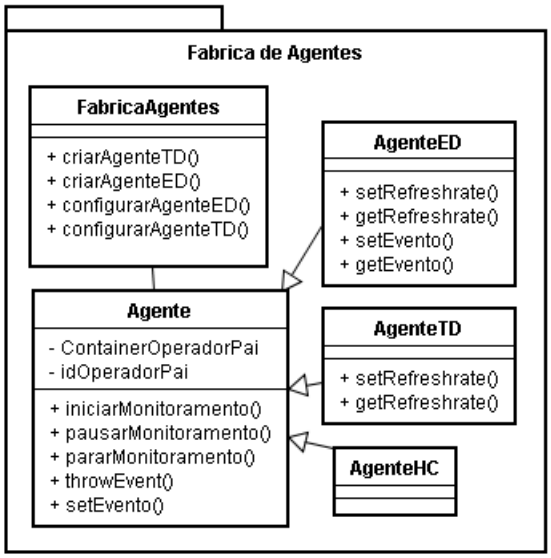

Figura 36. Pacote Fabrica de Agentes

5.4.7 Pacote Processamento de Consultas

A classe Evento (Figura 37) representa um evento simples, onde estão presentes os valores a serem comparados, o operador da comparação, e seu status atual. O tipo de operador é obtido diretamente do PEC. A classe Evento Simples é representada a seguir:

Um evento representa uma condição a ser monitorada pelo plano. As várias ocorrências do evento em pontos de tempo distintos são os *event reports*, associados com um conjunto de atributos A1, A2... An.

A classe Evento Composto (Figura 37) possui relacionamento 1:2 com a classe Evento Simples, ou seja, um objeto Evento Composto possui um ou dois eventos simples, além de operadores que calcula o resultado booleano baseado no *status* de cada um deles. A estrutura de árvore binária foi escolhida visto se adequar às representações tradicionais de planos de consulta.

A MEC é a responsável pelo monitoramento dos eventos compostos. É necessário o envio de mudanças no *status* do evento raiz para a MEC. A classe MEC relacional, inicialmente definida em [PINHEIRO, 2004], executa um PEC, supondo que as implementações de todos os operadores contidos neste plano estejam disponíveis. Atualmente, devido às novas demandas por funcionalidades de bancos de dados ativos, a MEC passa a ser um processo, que é executado até o término do plano de execução de consulta, onde a MEC aguarda os resultados serem enviados pelos agentes.

A classe Operador, na máquina de execução de consultas, representa os operadores enviados pela MEC para a detecção de eventos. Todos os operadores devem obedecer a uma mesma interface para que possam sem incluídos no plano de execução de consultas. Entretanto, existem algumas diferenças entre operadores unários, que só dependem de uma fonte (por exemplo, para monitoramento), e binários, que realizam *join* e outras operações entre duas relações.

Operadores simples possuem apenas um *resultset* como entrada, tendo apenas um operador como produtor associado. Todo o seu processamento é executado sobre este *resultset*. Os principais métodos são:

- executar(): é o método que obtém uma relação como resultado da operação, onde a lógica de processamento varia para cada operador.
- recebeMensagem(): este método atualiza o *resultset* do operador após o agente detectar um evento e enviar os dados relacionados.

Operadores binários necessitam de dois *resultsets* como entrada, oriundos de dois operadores produtores. Operações como *join* são executadas sobre estes *resultsets*.

A classe Container consiste em basicamente listas de operadores e agentes alocados a um nó na rede, e as informações sobre o conteúdo dos demais containers. Os pedidos de execução de operadores, e envio de *resultset* entre operadores passam pelos containers.

Os principais métodos são:

- receberNotificacao(): quando um agente ou operador é adicionado em algum container, todos os containers devem receber esta informação.
- executarOperador(): este método somente invoca o método executar() do operador, para comunicação entre containers.
- encaminharResultset(): como os operadores se encontram em nós distintos, deve ser feita a transferência da saída de um operador para seu consumidor. Este método realiza a transferência do *resultset* propriamente dita.
- A lógica com a decisão de qual container alocar um operador ou um agente está contida na MEC, e pode envolver consultas ao componente Metadados, Gerência de Fontes, entre outros. Este processo está intimamente ligado à otimização de consultas.

As classes *Resultset* e Tupla são recorrentes em projetos de integração de dados, representando uma tabela relacional.

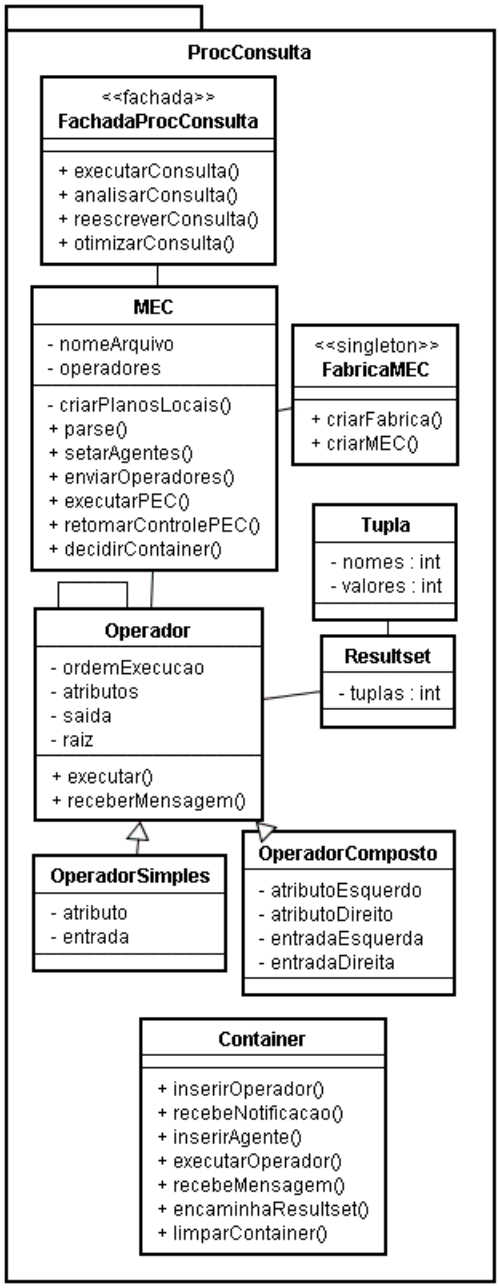

Figura 37. Pacote Processamento de Consulta

5.4.4 Pacote Gerência de Ocorrência (*event report*)

A classe Ocorrência (Figura 38) representa as ocorrências de eventos, simples ou compostos. Ela representa os horários de início do evento, os eventos intermediários que porventura ocorreram (no caso de eventos compostos), o horário de término e de disparo. O armazenamento de eventos passados (*event report*) pode ser utilizado para fins de consulta, processamento extra e descoberta de padrões. Eles representam visões históricas

da atividade do sistema.

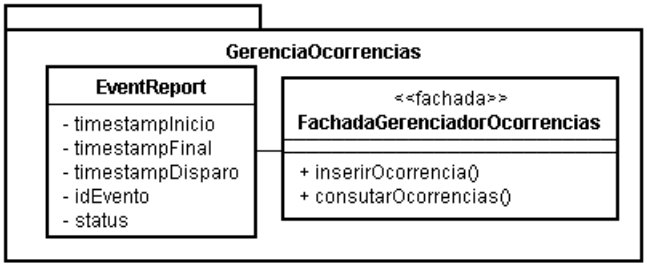

Figura 38. Pacote *Event Report*

Os principais métodos são:

- InserirOcorrencia(): insere o evento através do DAO
- recuperarOcorrencia(): recupera o evento por seu *id*.

5.4.5 Pacote Histórico

Este pacote (Figura 39) se refere à gerência de armazenamento definitivo do histórico contextual, não estando preso a limitações de *buffer*.

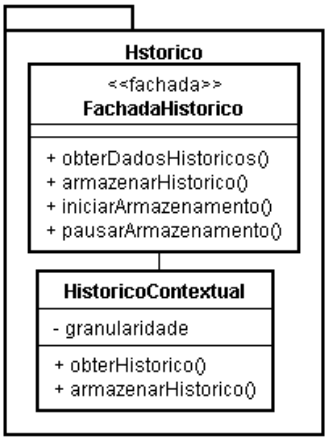

Figura 39. Pacote Histórico contextual

5.4.6 Pacote Perfil

Os dados de perfil (Figura 40) funcionam da seguinte maneira: quando um usuário realiza uma consulta, por exemplo, sobre a lista dos filmes que estão sendo exibidos. Durante a execução de consulta, serão verificados quais dados relativos a filmes estão presentes no perfil. Supondo que existe uma preferência sobre filmes de ação, a cláusula "*order by*

genero” será acrescentada ao plano. Caso contrário, a consulta não é alterada. Obviamente, deve-se verificar que seja feita uma correspondência correta entre o perfil e o esquema global presente nos metadados, o que é facilitado pelo uso de ontologias e realizado na fase de análise. A distinção entre dados de perfil e outros dados é justamente o usuário não precisar mencioná-lo explicitamente na consulta, o que torna a fase de reescrita mais apropriada.

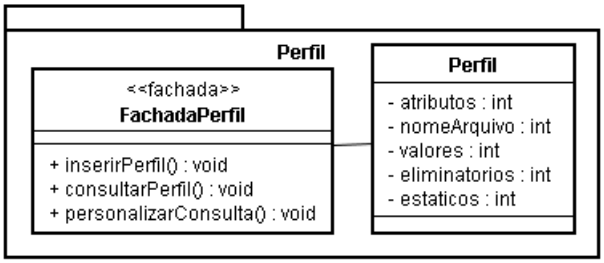

Figura 40. Pacote Perfil

Os métodos disponíveis são relativos às mudanças do coeficiente de preferência, criticidade (especificar se as tuplas que não atendam a este resultado devem ser eliminadas), além de informações sobre a duração daquela informação de perfil.

Obviamente, é necessário que os dados de perfil tenham correspondentes no esquema de metadados, para que uma correspondência seja feita, além de auxiliar no tratamento semântico de fontes heterogêneas de perfil [ROUSSOS; ZOUMBOULAKIS, 2004]. O principal método é:

- personalizarConsulta(): a partir de um PEC e um perfil de usuário, modifica o PEC incorporando as restrições e ordenações sobre atributos descritas no perfil.

5.4.7 Pacote Gerente de Tarefas

O módulo de gerência de tarefas é um dos mais importantes da arquitetura, definindo estruturas para a execução multi-usuários, realizando um novo conjunto de funcionalidades:

- Chamada ao módulo de tradução para a linguagem destino.
- Gerenciamento da fila de entrada e prioridades.
- Gerenciamento da fila de saída, de transações terminadas ou canceladas.
- Envio da subscrição para a *MECFactory*, incluindo o tipo de MEC a ser criada

pela fábrica.

Os principais métodos são:

- parserSubscricao(): obtém as informações de prioridade de autoria da subscrição. Não se trata da análise da consulta em si, que é tarefa do processador de consultas.
- checarMaiorPrioridade(): obtém a subscrição com maior prioridade no momento

As subscrições são requisições em XML enviadas através do módulo de gerência de subscrição da plataforma DBMWare. Esta classe acumula algumas funções auxiliares ao processamento de consulta, tais como gerar os planos locais de consulta e agentes.

A classe Prioridade (Figura 41) representa a prioridade de uma consulta em relação a outras, somente podendo ser modificadas por um administrador. A prioridade é recebida pela subscrição, no momento da inserção da consulta.

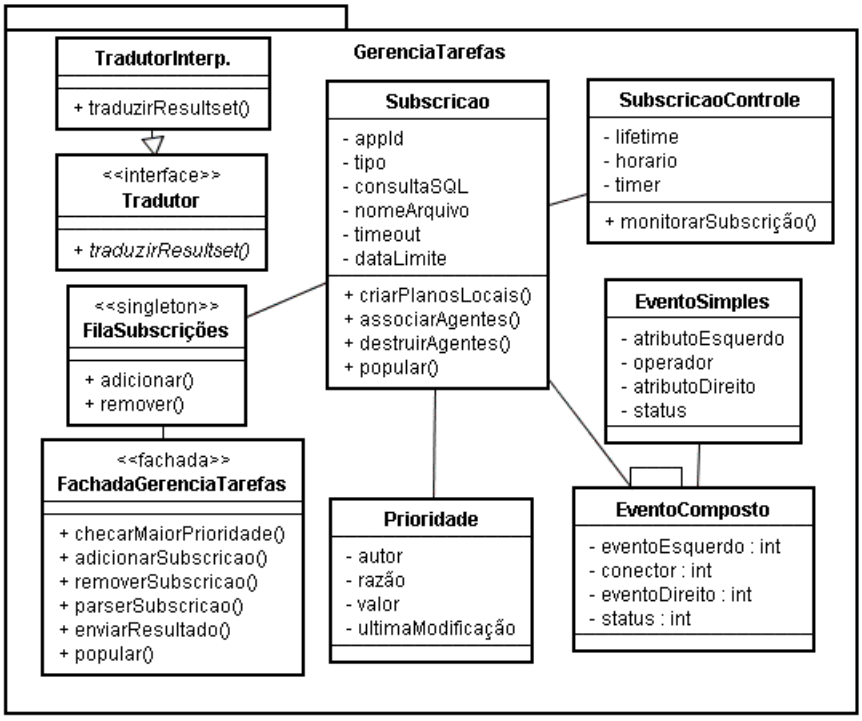

Figura 41. Pacote Gerente de Tarefas

A classe (Figura 42) Tradutor é responsável por compatibilizar o formato interno dos resultados da arquitetura, ao formato desejado pela aplicação requisitante. [OZSU; VALDIUREZ, 2001] o descreve como responsável pela interpretação dos comandos e

resultados ao usuário à medida que eles chegam. Os sistemas que possibilitam consultas em várias interfaces são chamados multilíngües. O exemplo ilustrado na Figura 43 se refere à tradução para nós representados em uma base de fatos OWL. Analogamente, os resultados podem ser entregues em outra linguagem necessária.

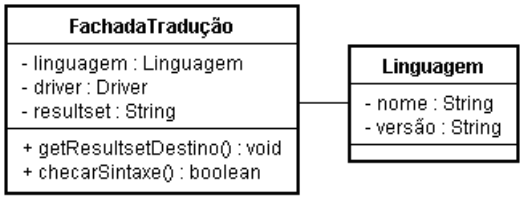

Figura 42: Classe Tradutor

A classe Linguagem armazena informações sobre a linguagem para a qual a tradução é feita, podendo ser utilizada para comparação quando vários tradutores forem oferecidos. Por exemplo, uma aplicação poderia especificar o formato XML para retornar o resultado e outra especificar objetos java serializados.

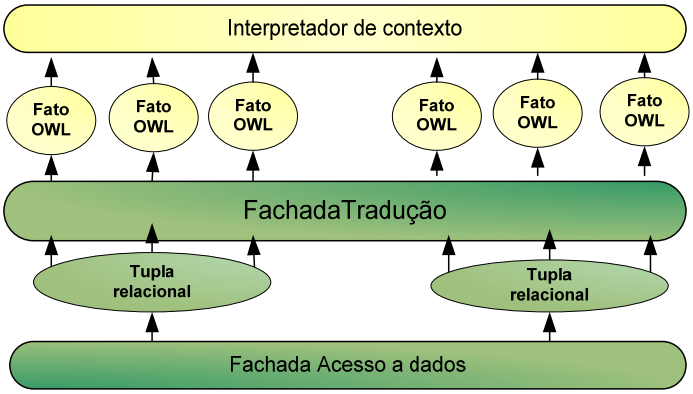

Figura 43: Funcionamento da tradução de conjunto resultado

5.4.8 Fábrica de MECs

A classe FabricaMEC é responsável pela criação e remoção de MECs para cada subscrição. Ela atende ao padrão de projeto *Factory Method* [GAMMA *et.al.*, 1995], onde existe a fábrica abstrata, as fábricas concretas, instâncias concretas e uma classe abstrata para o conceito de MEC. As MECs são inicializadas de acordo com cada consulta e a indicação do tipo de MEC é contida na mensagem enviada pelo gerenciador de tarefas.

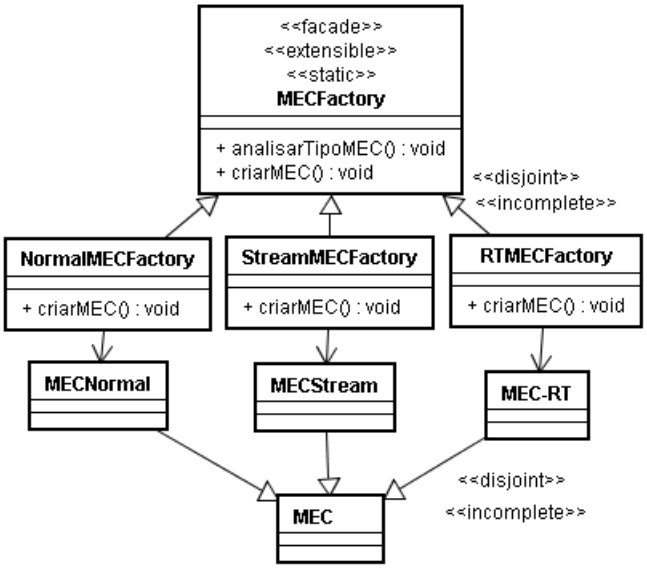

Figura 44. Fábrica de MECs

Conforme definida na UML-F, "*Disjoint*" significa que uma MEC só pode pertencer a um tipo, e "*incomplete*" significa que este conjunto inicial pode ser aumentado.

5.4.23 Pacote Gerência de fontes

A gerência de fontes está freqüentemente verificando o status das fontes, e alertando em casos de desconexão. No caso de o usuário fazer uma consulta *request-response* que não pode ser completa, ele deve receber um aviso em alto nível sobre esta impossibilidade. No caso de consultas *time-driven*, os horários de desconexão serão armazenados.

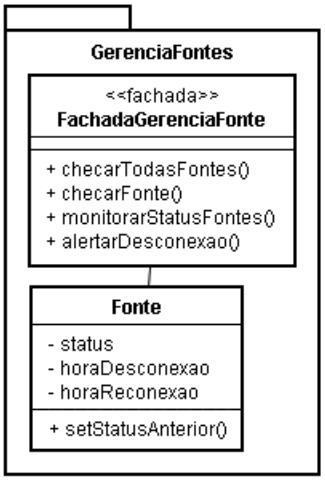

Figura 45. Pacote Gerente de Fontes

Os principais métodos são:

- checarStatusFonte(): verifica apenas uma fonte, uma vez
- checarTodosStatus(): fornece as situações de todas as fontes naquele momento
- monitorarTodosStatus(): verificação contínua das fontes
- alertarDesconexao(): quando uma fonte é desconectada, tal fato é armazenado

A classe Fonte (Figura 45) se refere às informações sobre as fontes de dados, por exemplo, autoria, tipo da fonte, IP e porta de acesso. Estas informações são obtidas através dos metadados. A grande distinção entre esta classe e a definida anteriormente em [SILVESTRE, 2005] é a presença de novos estados da fonte, tal como indisponível, além de métodos para o seu monitoramento.

Como várias fontes se encontram móveis ou com restrições de conectividade, o processador de consultas deve estar ciente do status atual das fontes. Os principais métodos são:

- checarStatus(): verifica o status de uma fonte apenas uma vez
- monitorarStatus(): equivale a checar o status da fonte repetidamente

5.4.10 Pacote DAO

A classe DAO (*Data Access Object*), como o nome indica, é uma implementação do padrão de projeto homônimo. Ele é responsável pela persistência dos diversos objetos mencionados ao longo deste capítulo.

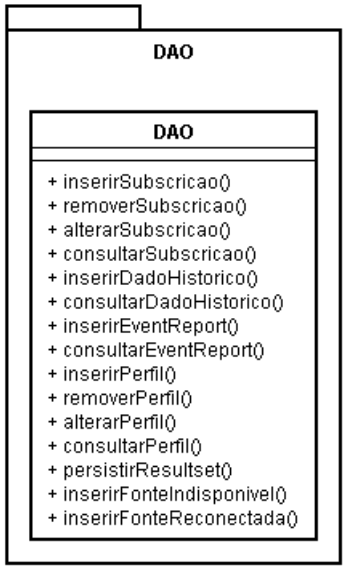

Figura 46. Pacote DAO

Uma vantagem do padrão DAO é desacoplar os objetos da camada de armazenamento de dados. Para este trabalho, foi escolhido o XML como formato padrão, mas futuramente qualquer formato pode ser utilizado: um exemplo seria o SGBD PostgreSQL, através do *Hibernate*, e a configuração pode variar para cada tipo de objeto.

5.4.11 Pacote Interfaces Gráficas

As subscrições são requisições em XML enviadas através do módulo de Gerência de tarefas. O CoDIMS pode ser utilizado através de chamadas a *web services*. Entretanto, para facilitar o uso do ambiente, foram criadas interfaces gráficas para os principais casos de uso, que podem ser vistos na Figura 56.

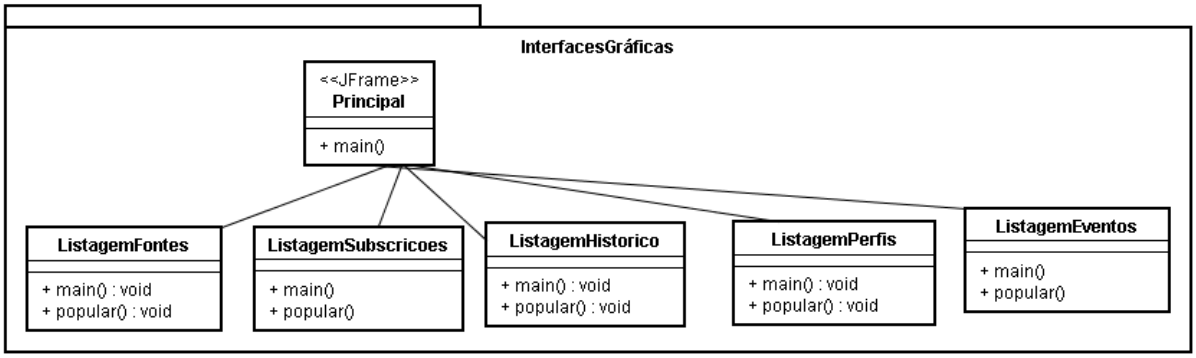

Figura 47: Pacote Interfaces Gráficas

5.5 Diagramas de Estado

Algumas transições de estado são relevantes para o sistema, como por exemplo, os estados de uma subscrição durante seu ciclo de vida, vistos na Figura 48. Assim que uma subscrição é criada e instanciada de acordo com seu tipo (*request-response*, TD ou ED), ela pode ser imediatamente atendida ou entrar em espera na fila. Quando ela for atendida é iniciada sua execução. Durante sua execução, ela pode ser interrompida quando uma subscrição com maior prioridade é inserida. Finalmente, a consulta é finalizada, ou pode ser cancelada pelo usuário.

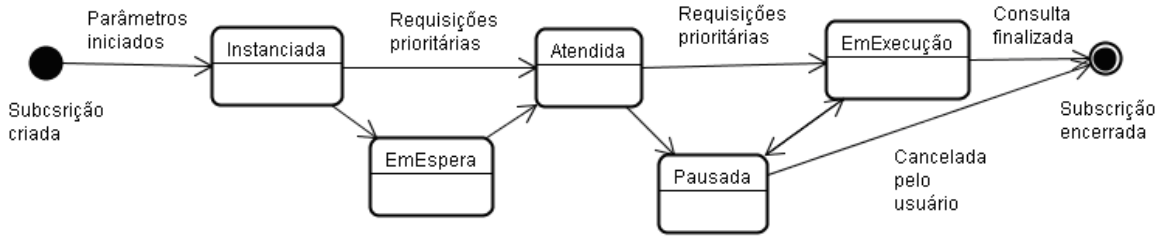

Figura 48. Diagrama de estados da classe Subscrição

Os *wrappers* e fontes de dados também podem mudar constantemente de estado, como mostrado na Figura 49. Após a criação de um *wrapper*, ele é disponibilizado para uso. A partir do estado acessível, ele pode ficar indisponível, caso aconteça algum problema no *web service*, ou a sua fonte de dados pode se tornar indisponível. Após a fonte de dados se tornar obsoleta, o *wrapper* é removido.

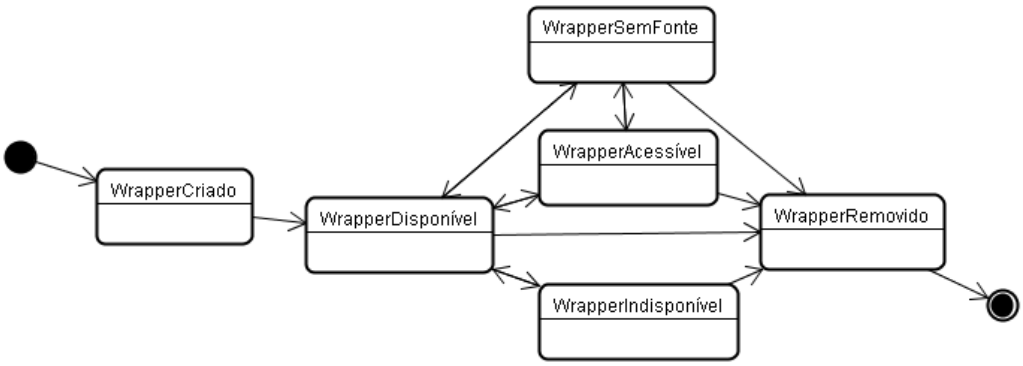

Figura 49: Diagrama de estados da classe *Wrapper*

5.7 Fluxo de dados

Para melhor ilustrar o funcionamento desta nova instância do CoDIMS, o CoDIMS-CA, será descrito o fluxo de processamento de uma subscrição, de acordo com a Figura 50. O usuário ou a aplicação requisita o monitoramento de um evento através de uma subscrição, definida em XML *(1)*. Cada subscrição possui identificador, tipo, *timeout* da consulta e prioridade, que pode ser modificada por um administrador do sistema *(2)*. As subscrições são enfileiradas no Gerenciador de Tarefas *(3)*. Para ser processada, a susbcrição é enviada ao Analisador que a valida, consultando os metadados, gerando um PEC. O PEC passa então pelos estágios de reescrita (agrupa as sub-consultas por fonte) e otimização. Uma MEC para execução deste PEC é criada, com modo de execução apropriado ao tipo da subscrição *(4)*. O PEC é então processado e reescrito *(5 a 10)*.

A MEC executa o PEC repassando as sub-consultas ao componente Acesso aos Dados, onde, para permitir comportamento ativo, agentes simples e compostos são criados *(12)*. O PEC é executado em uma seqüência onde alguns operadores podem ser executados paralelamente. A saída de um operador (por exemplo, uma seleção ou detecção de evento) é utilizada como entrada de outro operador.

Os agentes monitoram as fontes a fim de garantir a entrega de dados de forma ativa. O monitoramento pode ser pausado ou cancelado pelo usuário, ou terminar em um momento pré-estabelecido, com a ocorrência de um evento ou o tempo definido na consulta. Quando o evento da subscrição for detectado, uma notificação é realizada através da Gerência de Ocorrências *(14) (15)*, que é uma mensagem contendo informações tais como o horário do evento. Ela será armazenada em um repositório para ser consultada futuramente *(13)*.

Os conjuntos de dados retornados por cada fonte *(16)* são enviados para o Componente Acesso aos Dados, que as repassa à MEC para realizar a junção dos *resultsets* parciais *(17)*. O conjunto resultado final é então enviado para o gerente de tarefas. Finalmente, a tradução para o formato de destino é realizada *(18)* e a saída gerada é enviada para a aplicação requisitante (19). Todos os objetos gerados durante a execução são desalocados *(20)*.

Para cada um dos quatro modos de execução de consulta, a execução do PEC se dá de forma distinta: no *request-response* é enviada uma consulta tradicional à fonte de

dados; no *event-driven*, um agente para monitorar cada evento é criado; no *time-driven e event-driven*, um agente envia os dados nos tempos pré-determinados; Os agentes e os *wrappers* podem estar localizados fisicamente nas fontes de contexto, caso possível, senão remotamente.

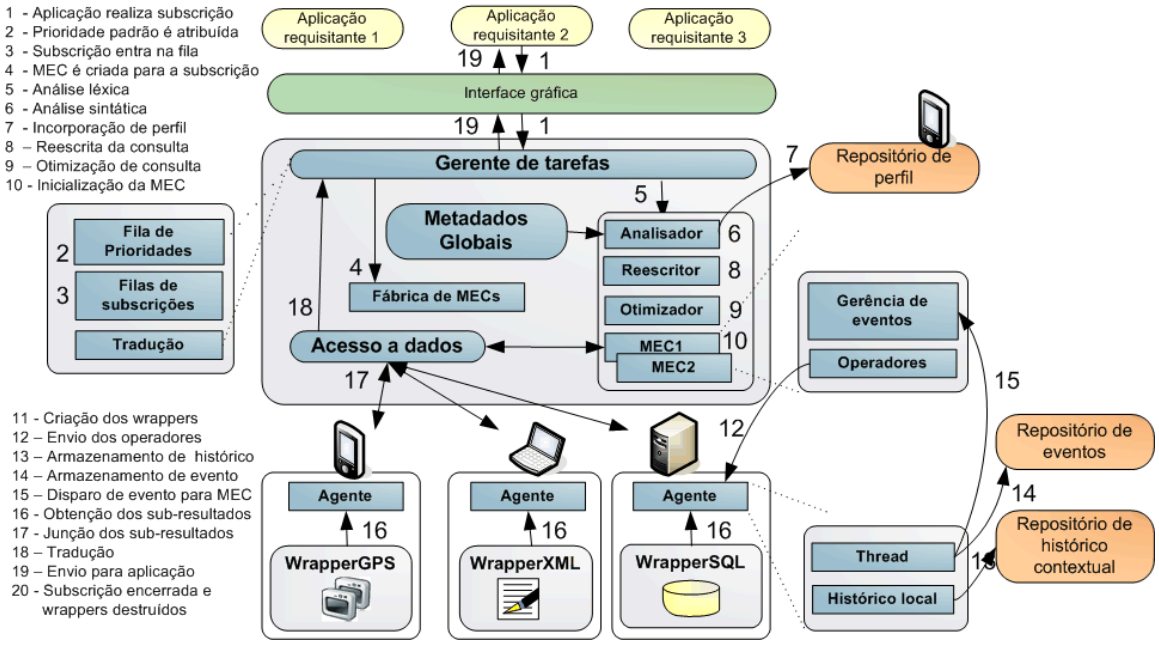

Figura 50. Fluxo de dados

Deve haver algum protocolo para a comunicação entre o módulo e os *wrappers* para o monitoramento de seu *status*, devido a freqüentes desconexões. Esta interação será realizada inicialmente com o ECGWrapper. [GONÇALVES, 2006]. Cada *wrapper* deverá retornar para a MEC o caminho referente ao arquivo de saída gerado e um código de retorno (Figura 51). Esse código de retorno se torna necessário pelo fato de o *wrapper* nem sempre conseguir retornar resultados.

```
0: resultado obtido com sucesso;
1: conjunto-resultado vazio;
2: fonte não conectada;
3: timeout;
4: fonte não localizada;
5: permissão negada
```

Figura 51. Retornos possíveis de um *wrapper* (extraído de [CÔCO, 2005]).

5.6 Diagramas de seqüência

Os diagramas de seqüência definem os fluxos de dados para as principais funcionalidades da arquitetura. O diagrama da Figura 52 define a configuração do *workfkow* do CoDIMS-

CA, onde os componentes a serem utilizados na instância são listados. Não há grandes mudanças em relação à definição de [BARBOSA, 2001], mas esta funcionalidade, no escopo da integração de dados contextuais, permite a incorporação de novos componentes desejados. Por exemplo, os componentes Agente e MEC distribuída fazem parte da configuração necessária do CoDIMS-CA, já a parte referente a estatísticas é opcional.

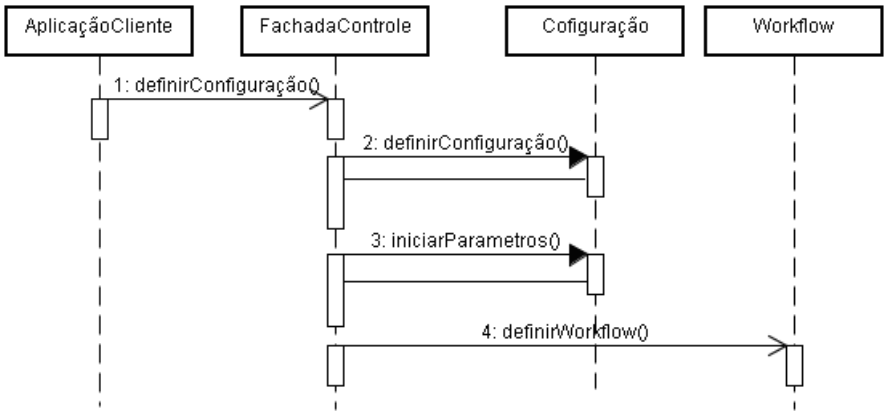

Figura 52. Seqüência de configuração do workflow (adaptada de [TREVISOL, 2004])

A Figura 53 representa a chegada de uma subscrição, sua entrada na fila de subscrições e execução, no caso de uso “Inserir Subscrição”, mostrado anteriormente.

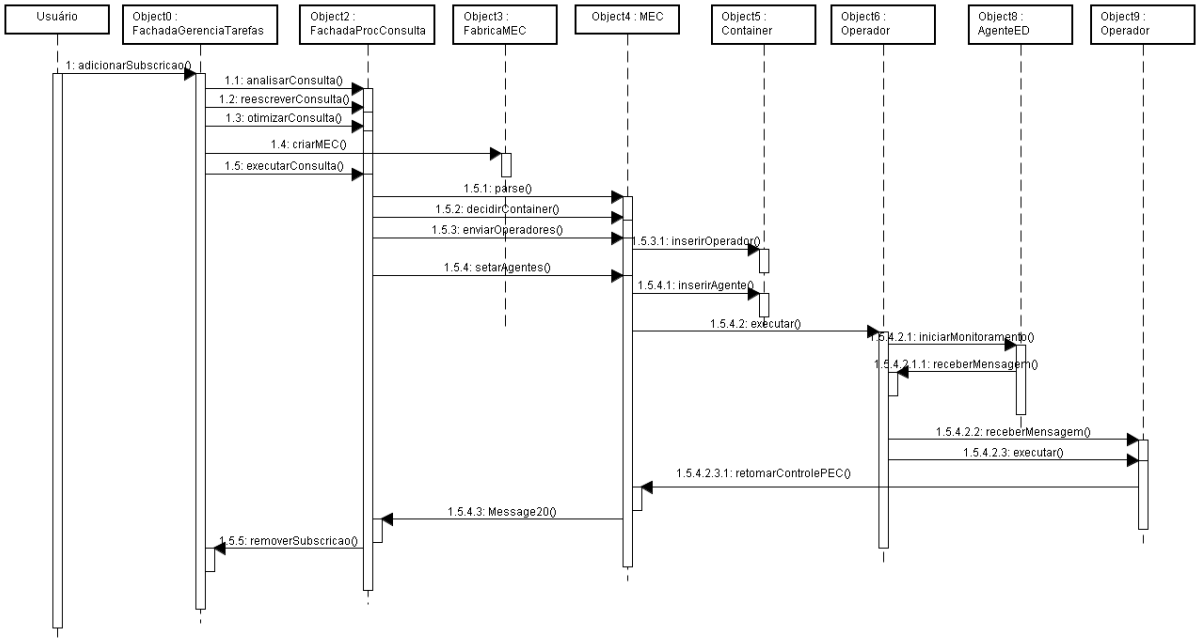

Figura 53. Seqüência de chegada de uma subscrição

O diagrama de seqüência da Figura 54 representa a execução de um plano de consulta *time-driven* pela MEC, com o acesso a vários operadores, como os de *loop* e junção (*loopNestedJoin*). Os demais operadores específicos da computação sensível ao contexto, como *aoAumentar*, também possuem esta interface.

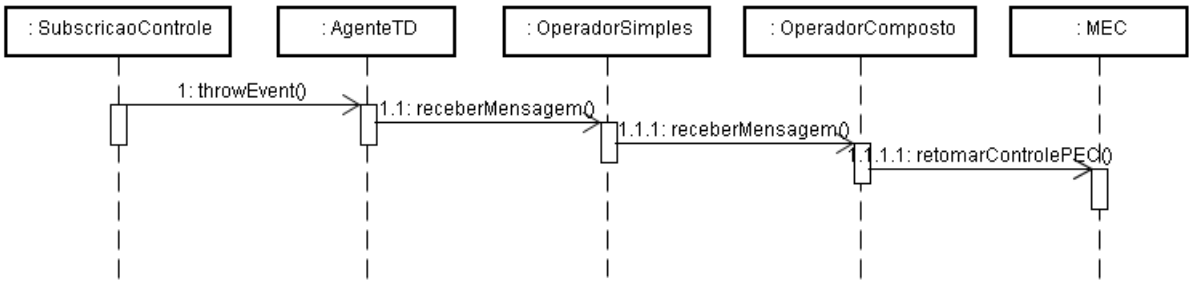

Figura 54. Seqüência de subscrição *time-driven*

A Figura 55 ilustra a detecção e armazenamento de ocorrência de eventos. A cada ocorrência de um evento simples, o agente registra a ocorrência e repassa ao evento composto, modificando seu status para “ocorrido”, havendo também um re-cálculo do *status* do evento composto. Quando todos os eventos são identificados, o evento composto tem seu *status* modificado, sendo disparadas então ações correspondentes.

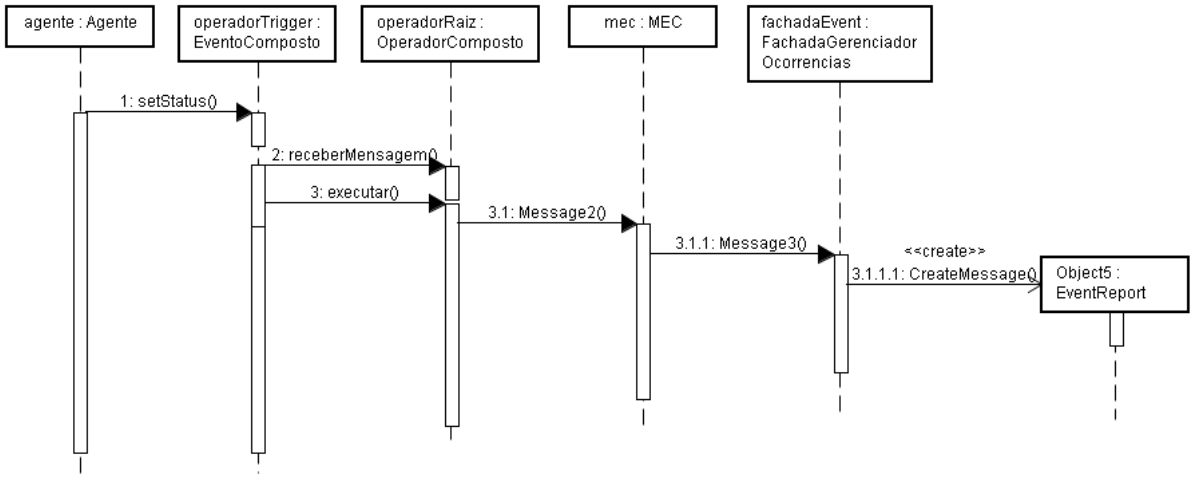

Figura 55: Seqüência do *event report*

O diagrama da Figura 56 representa a checagem das preferências de usuário na execução de um PEC, no caso de uso “Consultar Perfil”.

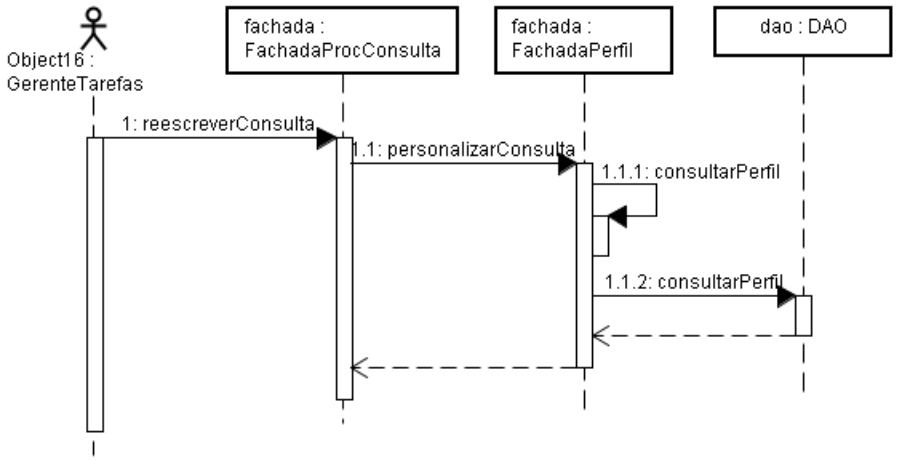

Figura 56. Sequência de verificação de perfil de usuário

O diagrama da Figura 57 mostra o monitoramento periódico do status das fontes. Pode-se fazer analogia com os modos de entrega de dados (*request-response*, *time-driven*, *event-driven*).

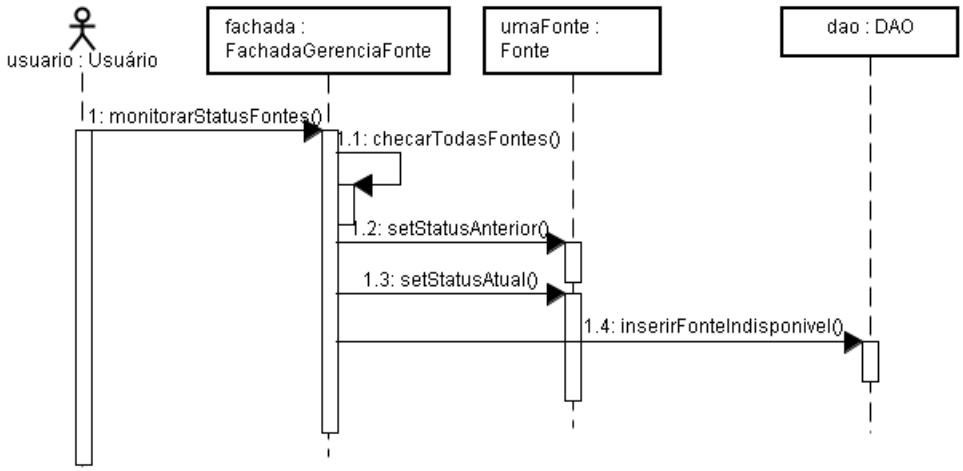

Figura 57. Sequência de gerenciamento de fontes

O diagrama da Figura 58 representa o armazenamento de um dado histórico, no caso de uso “Inserir histórico”. Primeiramente, o agente detecta a necessidade de armazenamento interno. Após cálculos necessários, através da hierarquia de contexto, a informação histórica é então armazenada no repositório centralizado.

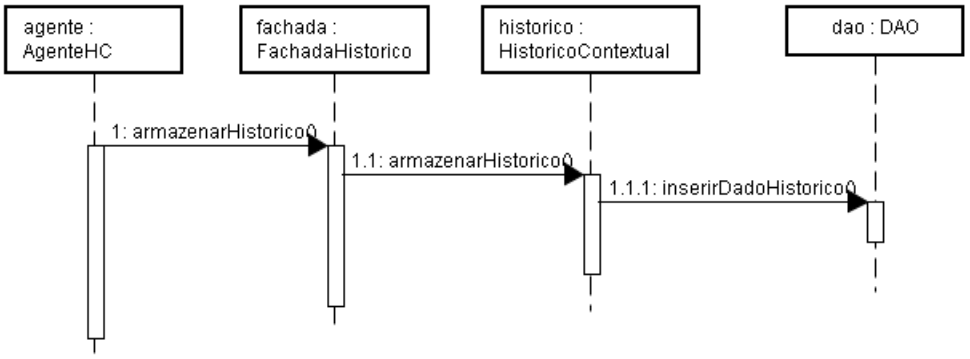

Figura 58. Sequência de armazenamento de histórico contextual

A figura 59 mostra um exemplo seqüência para uso das interfaces gráficas, desde a interface principal do sistema. Neste caso, a interface de listagem de subscrições cadastrada é acionada, e após a escolha do usuário, é realizado acesso ao DAO para a obtenção de detalhes da subscrição escolhida.

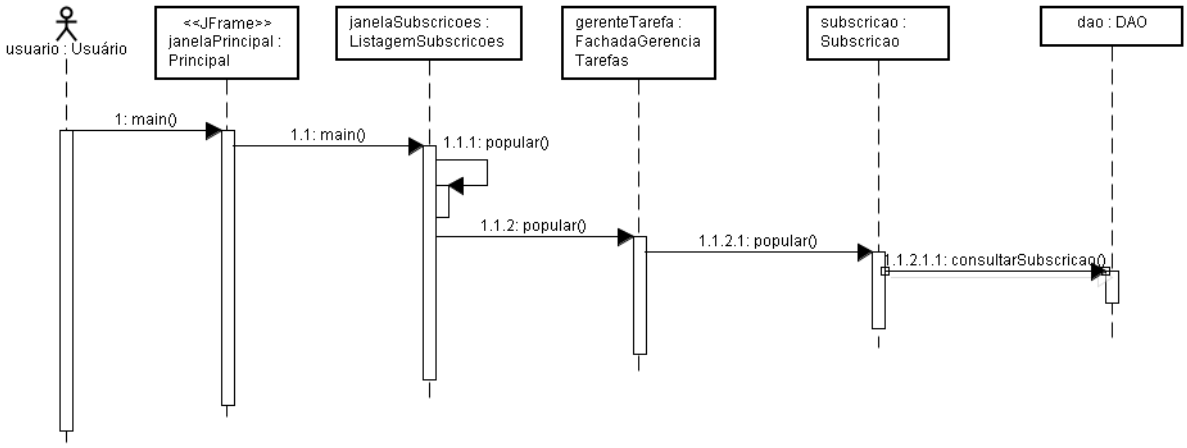

Figura 59: Sequência de uso de interfaces gráficas

Na Figura 60, é representada uma lista com os alguns operadores identificados: (*i*) operadores unários: possíveis de serem checados em uma única fonte; (*ii*) temporais: que podem ser absolutos em relação ao horário do sistema ou relativo a outro evento ocorrido; (*iii*) sobre transações de bancos de dados, tais como uma cláusula SELECT ou INSERT; (iv) de relação entre eventos, que possibilitam a comparação de diversos atributos. Extensões podem ser incluídas na arquitetura.

```
<xs:element name="NOT" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="onIncrease" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="onDecrease" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="onChange" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="equals" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="sum" minOccurs="0" maxOccurs="unbounded"/>
```

```
<xs:element name="count" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="repeat" minOccurs="0" maxOccurs="unbounded">
<xs:attribute name="times" type="xs:string"/>
<xs:attribute name="timeout" type="xs:string"/></xs:element>
<xs:element name="AND" minOccurs="0" maxOccurs="unbounded" />
<xs:element value="OR" minOccurs="0" maxOccurs="unbounded" />
<xs:element value="AafterB" minOccurs="0" maxOccurs="unbounded" />
```

Figura 60: DTD de operadores possíveis para detecção de eventos

Alguns operadores serão explicados por sua relevância:

- SCAN: este operador corresponde a um acesso simples
- ONINCREASE: verifica o acréscimo no valor de um atributo
- AND: operador booleano que verifica a veracidade de duas condições
- LOOP: repete uma operação de seleção periodicamente, no cenário *time-driven*

5.8 Metadados

Nesta seção é mostrada uma ontologia, baseada na ontologia criada em [SILVESTRE, 2005], responsável pela representação dos provedores de contexto. Propõe-se aqui sua integração futura com outras ontologias, sempre que possível e necessário. Alguns padrões potencialmente úteis para a descrição de fontes são o CC/PP e SensorML, descritos no Capítulo 2. Especificamente quanto aos metadados, a interface externa, disponível a aplicações e usuários, estes devem ser conceituais e independentes de tecnologias. Por sua vez, internamente, devem ser representados aspectos das fontes que são dependentes de tecnologias em particular. A ontologia completa é mostrada de maneira visual a seguir, na Figura 61.

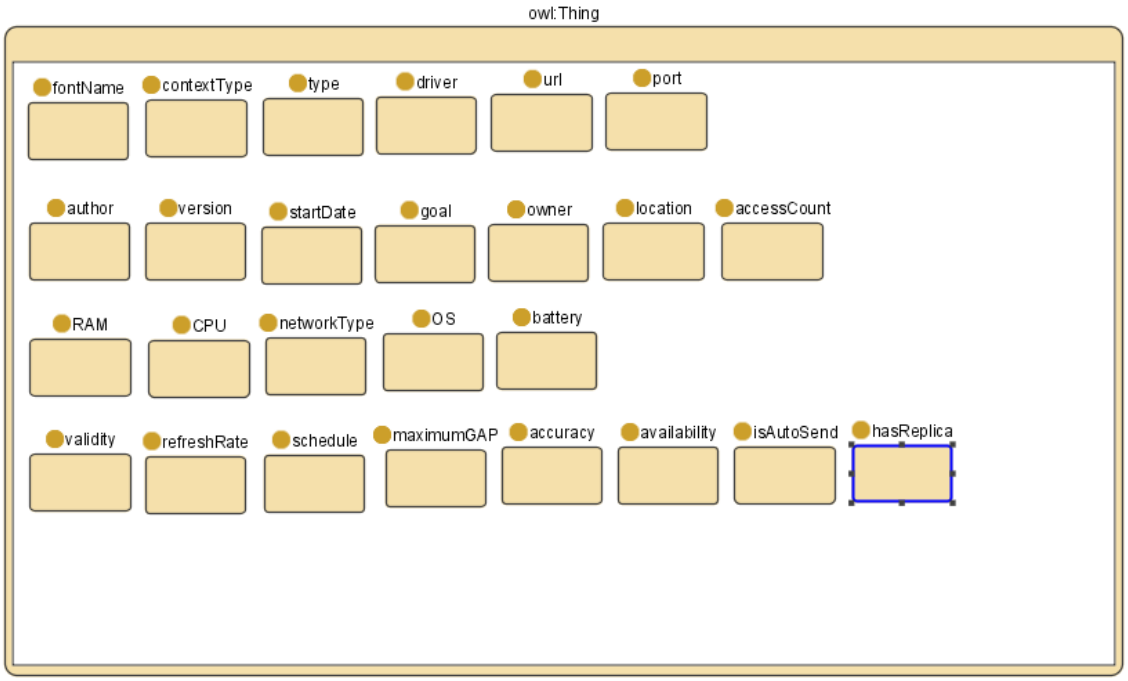

Figura 61: Descrição em OWL das fontes de dados

O código OWL correspondente a esta ontologia se encontra no anexo G

6 Estudo de caso

Neste capítulo será apresentada em detalhes a descrição do domínio da aplicação e suas funcionalidades. A aplicação requisitante também será descrita resumidamente. As interfaces com o usuário e o código-fonte implementado serão exibidos.

6.1 O domínio da telemedicina

Com os avanços na telemedicina e de aplicações médico-hospitalares, pode-se monitorar ininterruptamente e remotamente pacientes que têm doenças crônicas, sem a necessidade de internação, a partir de dados de sensores que podem ser transmitidos através de dispositivos móveis. Estes dados podem ser integrados com os dados de prontuários clínicos existentes em bancos de dados e repassados ao médico. A telemedicina também possibilita aos profissionais de saúde um melhor e mais preciso diagnóstico, proporcionando tratamentos mais rápidos em emergências e menores gastos governamentais.

Dentro da Telemedicina uma das áreas de destaque é a Telecardiologia, em particular o telemonitoramento da atividade cardíaca através do eletrocardiograma (ECG). O Telemonitoramento através do ECG tem despertado um grande interesse da comunidade científica devido ao alto índice de mortes associadas às doenças do coração, entre as quais podemos destacar a Isquemia do Miocárdio (IM). A IM é uma doença que se manifesta mais correntemente em condições normais de atividade do indivíduo e tem como conseqüência o infarto do miocárdio. Nessas condições, uma intervenção rápida e eficiente de uma equipe médica se faz necessária, a fim de garantir um tratamento médico mais eficaz, minimizando as conseqüências da doença.

No monitoramento domiciliar, o paciente é mantido em seu domicílio onde, por exemplo, seu sinal eletrocardiograma (ECG) é adquirido através de um aparelho *holter* e transmitido continuamente a um computador remoto que, por sua vez, envia os dados para a central de monitoramento; na ocorrência de uma emergência, mensagens de alerta são enviadas a unidade móvel solicitando assistência [MANTOVANELI, 2006].

Neste contexto, foi idealizado o projeto TeleCardio [PESSOA *et. al.*, 2006], cuja descrição é apresentada a seguir.

6.2 Projeto Telecardio

O projeto Telecardio tem como objetivo o monitoramento remoto de pacientes com risco de Isquemia no Miocárdio. A plataforma pode receber e enviar dados dos pacientes aos médicos responsáveis em função das mudanças de contexto.

Nos sistemas de telemedicina existentes na literatura, uma característica comum é que eles, geralmente, fornecem soluções isoladas e parciais de tele-monitoramento. A motivação do projeto Telecardio está no desenvolvimento de uma infra-estrutura diferenciada, em virtude da natureza heterogênea e distribuída dos ambientes em que estão inseridas as aplicações de tele-medicina. Uma outra motivação para a utilização de uma arquitetura flexível e configurável de telemedicina é que, segundo [FURUIE *et. al.*, 2003], as soluções adotadas para a área de saúde devem ser suficientemente flexíveis para que novos requisitos possam ser incorporados.

No caso particular do domínio de aplicação adotado neste trabalho (área de Saúde), a utilização da plataforma Telecardio fornece facilidades para a concepção de uma ampla variedade de aplicações de telemonitoramento, como a possibilidade de reuso de componentes, das interfaces padronizadas com sistemas de sensoriamento, das linguagens de subscrição, dos mecanismos de descoberta de serviços e das primitivas de manipulação das bases de armazenamento de dados biomédicos [PESSOA *et. al.*, 2006].

O TeleCardio é dividido em quatro camadas: *i)* a de aplicação médica, *ii)* interpretação de contexto (que engloba a interpretação de contexto, busca de serviços do Infraware), *iii)* integração e acesso a dados (contemplando a camada de acesso e integração de dados do Infraware), *iv)* fontes de dados (englobando os *wrappers*). Este trabalho é usado na camada de integração e acesso a dados, como mostra a Figura 62.

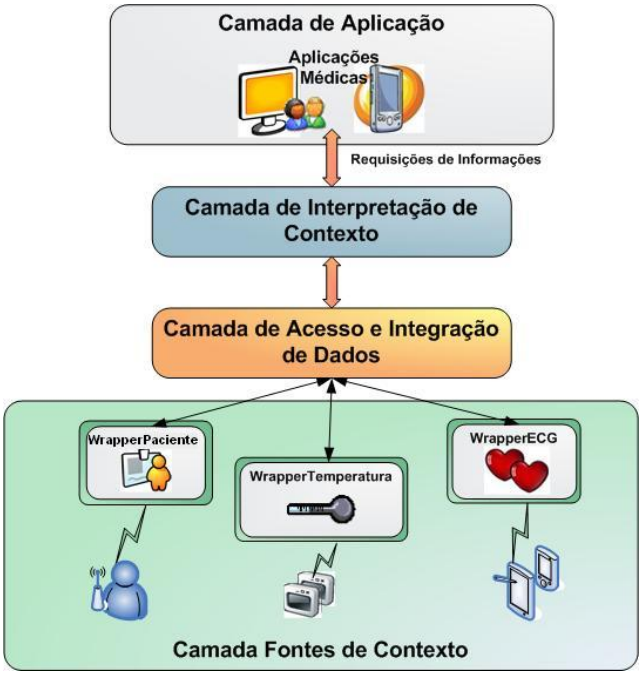

Figura 62. Arquitetura do Projeto Telecardio

6.2.1 Camada de fontes de contexto

Na Figura 62 nota-se a presença das fontes relativas aos sensores de temperatura e eletrocardiograma, além de um banco de dados de pacientes (prontuário). Exemplos de dados de pacientes são o histórico de diabetes, hipertensão, medicamentos indicados e internações anteriores. Em especial, neste cenário são importantes as informações de eletrocardiograma, representadas por um *Wrapper* ECG.

O Módulo de Processamento de Sinal ECG [ANDREAO *et. al.*, 2006] está organizado em uma estrutura composta por três camadas (Figura 63), não sendo, entretanto, o foco deste trabalho. Na ocorrência de um evento anormal detectado pelas Camadas 1 e 2 do referido módulo, o *Wrapper* ECG é notificado e envia a informação de ocorrência desse evento à central de monitoramento.

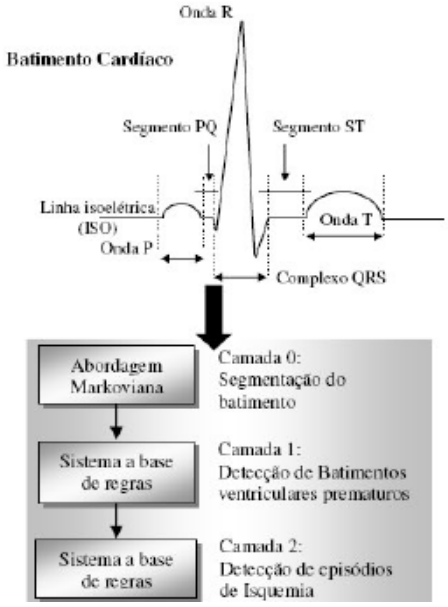

Figura 63: Módulo de Processamento de Sinais ECG [ANDRIAO *et.al.*, 2006]

Para o monitoramento cardíaco, deve ser provida a visualização simultânea de dois canais de sinal de ECG ambulatorial previamente armazenado em trechos de 10 segundos, assim como os resultados dos algoritmos que fazem a segmentação do sinal, analisando e identificando as diferentes formas de ondas que compõe o ECG. Os eventos gerados pelos algoritmos de classificação também são observados em janelas gráficas destacando o evento desejado. Para o caso das funções intervalo R-R (serie temporal da distância entre dois batimentos consecutivos) e desvio de segmento ST ao longo do tempo, o sinal observado difere do sinal ECG devido e requer uma escala temporal mais longa e de menor resolução. A exibição do intervalo R-R possibilita acompanhar a elevação e a queda da freqüência cardíaca instantânea (quanto maior o intervalo R-R mais baixa é a freqüência cardíaca e vice-versa).

6.2.2 Camada de acesso e integração de dados

A Camada de acesso se encontra entre a Camada de fontes de dados e a Camada de interpretação de contexto. O CoDIMS-CA realiza a função de acesso e integração de dados para a plataforma Telecardio. Entretanto, seu projeto configurável permite que outros ambientes de telemedicina com as mesmas necessidades possam ser utilizados.

6.2.3 Camada de Interpretação de Contexto

A camada de interpretação de contexto, apresentado em [PESSOA, 2006], é baseado na linguagem OWL, o que permite maior expressividade semântica. Esta decisão de projeto possibilita a realização de inferências facilmente, através da própria máquina de inferência contida no *framework* Jena, além do uso de ontologias padrões para cada domínio (no estudo de caso, o domínio de tele medicina). Sua arquitetura é vista na figura 64.

Figura 64: Interpretador de contexto (extraído de [MANTOVANELI, 2006]).

Neste interpretador, informações contextuais são representadas através de uma base de regras e fatos. A partir de regras, inferências podem ser feitas, dados de mais alto nível podem ser obtidos e serviços podem ser chamados para executarem tarefas (como por exemplo, a notificação de médicos). Este interpretador é a aplicação requisitante de nosso cenário. Um exemplo de código pode ser visto a seguir, na Figura 65:

```
<owl:Class rdf:about="Paciente">
<rdfs:label>Paciente</rdfs:label></owl:Class>

<owl:ObjectProperty rdf:about="estaLocalizadoEm">
<rdfs:domain rdf:resource="#Paciente"/>
<rdfs:type rdf:resource="&owl;FunctionalProperty"/>
<rdfs:range rdf:resource="#Lugar"/>
</owl:ObjectProperty>
```

Figura 65: Regra do interpretador de contexto

A interpretação de contexto é uma das fases da execução de sistemas sensíveis ao contexto, segundo [TOMASIC; SIMON, 1997]. O resultado da interpretação de contexto são os dados derivados repassados à aplicação.

6.3 Descrição do cenário

O cenário descrito como exemplo, definido em conjunto com as partes envolvidas no Telecardio, constitui de um dispositivo portátil que envia blocos de dados de 30s (250

amostras/segundo de 12 bits cada) até um computador remoto (*desktop*) através de uma interface de comunicação sem fio. O computador remoto por sua vez se comunica com um servidor via Internet o qual armazena os dados em um prontuário do paciente. O servidor possui uma aplicação *web* com uma lista de pacientes em monitoramento. Esse monitoramento se prolongará por dias. O plano de consulta utilizado pelo módulo, e criado no momento da subscrição, especifica que a consulta será executada até que algum usuário ou administrador do sistema a cancele.

Durante o monitoramento, o paciente aciona um botão de alarme localizado no dispositivo portátil, ou os sensores capturam condições de risco. Essa ação deve ser tratada com prioridade máxima por parte do operador da central de monitoramento. Neste caso de emergência, o modo de execução para este tipo de emergêcia é preemptivo, ou seja, a Camada de acesso e integração iniciará imediatamente o processamento desta consulta, sobrepujando outras subscrições (como pedido de um relatório administrativo) em fases como o retorno dos resultados à aplicação.

Em outra ocasião, o médico deseja visualizar constantemente os sinais vitais através de uma interface gráfica. Este caso representa uma subscrição *time-driven*, que pode ter um nível de prioridade mais baixo.

Os atores para estes cenários, segundo [EKKEBUS, 2006], são:

• Profissional de saúde: o profissional de saúde monitora o paciente e é diretamente envolvido com o tratamento deste;

• Desenvolvedor de aplicações: responsável por implementar um cenário específico e realizar ajustes no ambiente. Inclui integração com a infra-estrutura do hospital.

6.4 Descrição da aplicação

Uma aplicação deseja monitorar um paciente através de seu ECG e sua temperatura. Caso a freqüência cardíaca (FC) esteja maior que 90 e crescente, a temperatura esteja maior que 39ºC e crescente, e o paciente seja fumante, o médico deve receber um alarme e os últimos 10 dados de FC obtidos, além do identificador, nome e idade do paciente. Na figura 66 é apresentada, na linguagem *SQL-like* adotada, a requisição em questão, enviada pela Camada de Interpretação de Contexto à Camada de Acesso e Integração de Dados (CoDIMS-CA).

```
SELECT p.idPaciente, p.nome, p.idade, t.temperatura last 10 e.FC values
FROM paciente p, ecg e, temperatura t
WHEN e.FC onIncrease AND t.temperatura onIncrease
```

```
e.FC > 90 AND t.temperatura > 39 AND
p.fumante = “sim”
```

Figura 66: Subscrição em formato *SQL-like*.

Para atender a esta requisição é necessário acessar e integrar dados de diferentes fontes apresentadas na Tabela 21.

Tabela 21: Dados das fontes envolvidas na consulta

Fonte de Dados	**Nome**	**Origem dos dados**	**Tipo dos dados**
Registros de Pacientes (idPaciente,nome,idade,sexo,fumante)	Prontuário	Servidor do hospital	Relacional
ECG de pacientes (idPaciente, frequencia, data-hora)	ECG	Sensor no paciente	XML
Temperatura dos pacientes (idPaciente, valor, data-hora)	Temperatura	Sensor no paciente	TXT

A Figura 67 mostra os esquemas de criação das fontes em seus formatos originais.

```
<DOCTYPE >
<!ELEMENT ECG >
<!ATTLIST ECG
       idPaciente          #REQUIRED
       frequencia          CDATA
       data-hora           CDATA>
```

```
Temperatura
          IdPaciente
          Temperatura
          Data_hora
```

```
CREATE TABLE  Pacientes
       id           integer(9)   #PK
       nome         varchar(90)
        sexo         integer(1)
       idade        integer(3)
       fumante      integer(1)
```

Figura 67: Esquemas internos das fontes de dados

Como descrito anteriormente, o CoDIMS-CA é baseado em componentes e implementados como serviços Web (Web services), o que simplifica o seu acesso por diversas aplicações. Assim, inicialmente, torna-se necessário configurar os componentes do CoDIMS-CA para o TeleCardio, com a incorporação dos novos componentes em relação ao CoDIMS original (Figura 68).

```
<configuracao><hosts>
<nome_comp=”Controle” host=”127.0.0.1” porta=”9090” uri=”urn:controle”>
<nome_comp=”AcessoDados” host=”127.0.0.1” porta=”9090” uri=”urn:acessodados”>
<nome_comp=”MetaDados” host=”127.0.0.1” porta=”9090” uri=”urn:metadados”>
<nome_comp=”ProcConsulta” host=”127.0.0.1” porta=”9090” uri=”urn:procconsulta”>
<nome_comp=”Agente” host=”127.0.0.1” porta=”9090” uri=”urn:agente”>
<nome_comp=”Historico” host=”127.0.0.1” porta=”9090” uri=”urn:historico”>
<nome_comp=”EventReport” host=”127.0.0.1” porta=”9090” uri=”urn:evenTReport”>
<nome_comp=”GerenciaTarefas” host=”127.0.0.1” porta=”9090”
uri=”urn:gerenciaTarefas”>
<nome_comp=”Container” host=”127.0.0.1” porta=”9090” uri=”urn:container”>
</hosts></configuração>
```

Figura 68: Componentes instanciados para cenário do TeleCardio

Para iniciar a execução de uma consulta é executado um *Web service*. A chamada para este *Web service* pode ser visto no trecho de código da Figura 69, que invoca uma ação do componente Controle. Outras ações do sistema também são invocadas via *Web services*.

```
/* configura o componente Controle,de acordo com o arqConfiguracao.xml */
    call.setTargetObjectURI("urn:controle");
    call.setMethodName("definirConfiguracao");
    call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
    Vector params = new Vector();
    params.addElement(new Parameter("arquivo", String.class,
"C:/CoDIMS/Configuracoes/arqConfiguracao.xml", null));
    call.setParams(params);
    Response response = call.invoke(new
URL("http://localhost:9090/soap/servlet/rpcrouter"), "");

    call.setMethodName("exibirConfiguracao");
    call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
    params.removeAllElements();
    response = call.invoke(new URL("http://localhost:9090/soap/servlet/rpcrouter"),
```

Figura 69: Exemplo de execução de *web service*

O próximo passo é a definição dos metadados das fontes de dados envolvidas no CoDIMS-CA. Os metadados da fonte de dados ECG deste cenário são descritos na Figura 70:

```
<owl:FunctionalProperty rdf:ID="fontName">Holter</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="type">
<type rdf:about="#ECG"/></owl:FunctionalProperty>
<owl:DatatypeProperty rdf:ID="url">10.0.0.11</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="port">9090</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="driver">WrapperHolter.clas</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="refreshRate">30000</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="isAutoSend">yes</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="maximumGap">500</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="acuracy">2</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="contextType"><owl:oneOf>
<contextType rdf:about="#Sensed"/><owl:oneOf/> </owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="author">Rodrigo Varejão</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="owner">DEE-UFES</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="goal">Detectar variações que indiquem IM e outras
complicações, através de algoritmos</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="OS"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="RAM"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="CPU"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="battery">12</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="location"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="network"><owl:oneOf rdf:parseType="Collection">
<contextType rdf:about="#WI-FI"/></owl:FunctionalProperty>
```

Figura 70: Metadados da fonte ECG

Os dados sobre a fonte de dados acima significam que a fonte é atualizada a cada 30 segundos e automaticamente (o fabricante realiza a gravação no próprio sensor); pode haver um atraso de no máximo 500 milisegundos, e inexatidão no valor de até 2 bpm.

```
<owl:FunctionalProperty rdf:ID="fontName">Paciente</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="type">
<type rdf:about="#Paciente"/></owl:FunctionalProperty>
<owl:DatatypeProperty rdf:ID="url">10.0.0.12</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="port">9090</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="driver">WrapPaciente.clas</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="isAutoSend">no</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="hasReplica">Yes</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="contextType"><owl:oneOf>
<contextType rdf:about="#Primitive"/><owl:oneOf/> </owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="author">Rodrigo Varejão</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="owner">DEE-UFES</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="goal">Detectar variações que indiquem IM e outras
complicações, através de algoritmos e redes de Markov</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="startDate">2007-08-20</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="OS">Suse 10</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="RAM">2048Mb</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="CPU">2.3</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="location"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="network">Ethernet 100Mb<owl:oneOf
rdf:parseType="Collection">
<contextType rdf:about="#WI-FI"/></owl:FunctionalProperty>
```

Figura71: Metadados da fonte Paciente

Como os dados da fonte paciente não são provenientes de sensores, algumas propriedades não são aplicáveis, como taxa de atualização e atraso. O tipo de dados é “primitivo” e esta fonte possui réplica. As diferenças no poder de computação são nítidas. A fonte temperatura, por sua vez, não é reproduzida aqui, sendo similar à fonte ECG. A Tabela 22 mostra a distribuição dos componentes para esta aplicação.

Tabela 22: Componentes disponíveis

URL	**Componentes**
10.0.0.1	wrapperPatient, container1, dataAccess, QEE
10.0.0.2	wrapperECG, container2, metadata, dataSourceManager
10.0.0.3	wrapperTemperature, container3, eventReport, historic

6.5 Fluxo de dados

Para execução da subscrição apresentada na Figura 66, é necessário seguir todo o fluxo descrito anteriormente na seção 5.7. Assim, a partir da subscrição é gerado um plano de execução de consulta (PEC), (Figura 72), consistindo dos seguintes passos:

1) Análise léxica: são verificados os erros de escrita, tais como palavras-chave escritas incorretamente, ponto-e-vírgula;

2) Análise sintática: as instruções *SQL-Like* são efetivamente convertidas em operações de um plano de consulta: a instrução SELECT gera operações de projeção e seleção; cláusulas

WHEN geram operadores baseados em eventos; cláusulas FROM geram junção e/ou composição de eventos;

3) Análise semântica: neste estágio os metadados são amplamente utilizados. São verificados se todos os atributos e tabelas existem e se não há nenhuma inconsistência. Um exemplo de inconsistência é criar operação baseadas em evento para atributos classificados como estáticos;

```
<PEC>
<operator execution-step ="1">  <Scan table="Temperature">
    <column name="id" /> <column name="value"/>
    <constraint leftArg=”value” operation=”bigger” rightArg=”39”/>
    <constraint OnIncrease  column name="value"/>
    <consumer id="3"/> </scan> </operator>
<operator execution-step ="2"> <Scan table=”ECG”>
   <column name="id" /> <column name="FC"/>
   <constraint leftArg=”FC” operation=”bigger” rightArg=”90”/>
    <constraint OnIncrease  column name="FC"/>
    <consumer id="3"/> </scan> </operator>
<operator execution-step ="3"> <And left-producer="1"   right-producer="2">
    <constraint leftArg="id" operation="equals" rightArg="id"/>
     <consumer id="5"/></And></operator>
<operator execution-step ="4"> <Scan table="Patient ">
     <column name="id" />
     <column name="age” />
    <column name="name "/>
     <constraint leftArg=”smoker” operation=”equals” rightArg=”yes” />
     <consumer id=”5”/> </scan></operator>

<operator execution-step="5"> <Join  left-producer="3"   right-producer="4">
     <constraint leftArg="id" operation="equals" rightArg="id"/>
     </ join></operator>

<operator execution-step="5"> <Join  left-producer="3"   right-producer="4">
     <constraint leftArg="id" operation="equals" rightArg="id"/>
     </ join></operator>

<operator execution-step ="6">  <Scan table="ECG">
    <column name="id" /> <column name="value"/>
    <constraint leftArg=”value” operation=”last” rightArg=”10”/>
    <consumer id="3"/> </scan> </operator>

<operator execution-step="7"> <Join  left-producer="5"   right-producer="6">
     <constraint leftArg="id" operation="equals" rightArg="id"/>
     </ join></operator>

</PEC>
```

Figura 72. PEC gerado a partir da subscrição

Este mesmo PEC pode ser representado graficamente através da árvore da Figura 73.

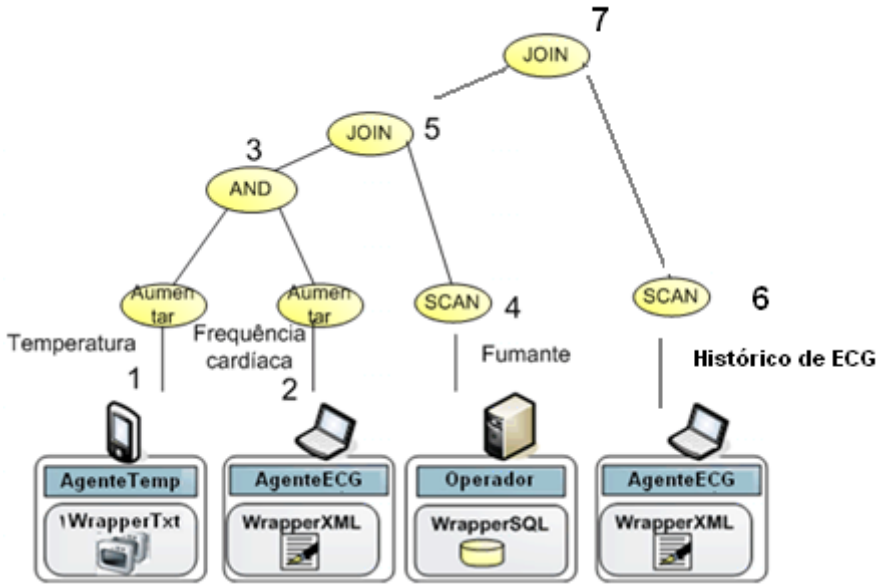

Figura 73. Visualização gráfica de PEC de monitoramento

O PEC é então executado pela Máquina de Execução de Consultas (MEC) previamente alocada para este PEC, encaminhando as sub-consultas ao componente Acesso aos Dados para serem executadas em cada fonte de dados. Os agentes necessários ao monitoramento de eventos são criados através de uma fábrica de agentes. No caso da sub-consulta para temperatura (Figura 74) um agente monitora a condição “se temperatura > 39”. De forma análoga, outro agente é criado para se monitorar a freqüência cardíaca. O acesso à fonte Paciente é do modo *Request-Reponse*.

```
<local-plano>
   <coluna ordem="0" nome="idPaciente" tabela="Temperatura" />
   <coluna ordem="1" nome="temperatura" tabela="Temperatura" />
      <tabela nome="Temperaura" />
<condição><argumentoEsquerdo nome=”temperatura”/>
< operador nome=”maior”/> <argumentoDireito nome=”39”/> <condição>
</local-plano>
```

Figura 74: Sub-consulta para temperatura

A Figura 75 mostra o resultado após a execução de cada sub-consulta (*resultsets*) bem como os dados repassados aos operadores de acordo com o PEC da Figura 72:

```
   Resultset do operador Scan: Temperatura
   <Resultset xmlns="http://www.w3schools.com">
   <tupla xmlns="" id_paciente="00001" frequencia="92"/>
   <tupla xmlns="" id_paciente="00002" frequencia="62"/>
   <tupla xmlns="" id_paciente="00003" frequencia ="91”/>
   <tupla xmlns="" id_paciente="00004" frequencia ="67"/>
   <tupla xmlns="" id_paciente="00005" frequencia ="71"/>
   <tupla xmlns="" id_paciente="00006" frequencia ="78"/></Resultset>

   Resultset do operador Scan: frequência cardíaca
```

```
   <Resultset xmlns="http://www.w3schools.com">
   <tupla xmlns="" id_paciente="00001" temperatura="39" />
   <tupla xmlns="" id_paciente="00002" temperatura="39" /></Resultset>

   Resultset do operador And:
   <Resultset xmlns="http://www.w3schools.com">
   <tupla xmlns="" id_paciente="00001" frequencia="92"/>
   <tupla xmlns="" id_paciente="00003" frequencia ="91"/></Resultset>

   Resultset do operador Scan: paciente
   <Resultset xmlns="http://www.w3schools.com">
   <tupla ordem="1" idPaciente="00001" nome="José da Silva" idade="52"/>
   </Resultset>

   Resultset do operador Join:
   <Resultset xmlns="http://www.w3schools.com">
   <tupla xmlns="" id_paciente="00001" temperatura="39" frequencia="92" idade="52"
sexo="masculino" fumante="sim /></Resultset>
   </Resultset>


   Resultset do operador Scan (dados históricos):
   <Resultset xmlns="http://www.w3schools.com">
   <tupla ordem="1" idPaciente"="00001" freq_historico_01="92" freq_historico_02="62"
                                    freq_historico_03 ="91" freq_historico_04 ="67"
                                    freq_historico_05 ="71"  freq_historico_06 ="78"
                                    freq_historico_07 ="88"  freq_historico_08 ="89"
                                    freq_historico_09 ="89"  freq_historico_10 ="92"/>
</Resultset>

  Resultset do operador Join (prontuário e histórico):
   <Resultset xmlns="http://www.w3schools.com">
  <tupla ordem="1" nome="José da Silva" idPaciente="00001" idade"="47" temperatura="39"
                                     freq_historico_01="92" freq_historico_02="62"
                                    freq_historico_03 ="91" freq_historico_04 ="67"
                                    freq_historico_05 ="71"  freq_historico_06 ="78"
                                    freq_historico_07 ="88"  freq_historico_08 ="89"
                                    freq_historico_09 ="89"  freq_historico_10 ="92"/>
</Resultset>
```

Figura 75: Seqüência de operadores e Resultado final

O interpretador de contexto do Telecardio, que é a aplicação requisitante, recebendo o conjunto resultado final, realiza a chamada a serviços correspondentes ao evento detectado, no caso, o disparo de mensagens para um celular.

6.6 Interfaces com o usuário

Várias interfaces gráficas foram criadas para facilitar e demonstrar o uso do CoDIMS-CA. As principais são:

a) Interface principal:

Figura 76: Interface principal do sistema

Menu Gerência Tarefas:

- Consultar subscrições
- Adicionar subscrição
- Remover subscrição
- Listar prioridades

Menu Histórico

- Listar dados históricos
- Armazenar histórico: configura e inicia a gravação de histórico para uma fonte

Menu Perfil

- Adicionar perfil
- Remover perfil
- Consultar perfil

Menu *Event Report*

- Listar eventos ocorridos

Menu Gerência Fontes

- Consultar status de fontes

b) Interface gráfica para listagem de pacientes: O conteúdo das demais fontes de dados, tais como ambulâncias e médicos, também pode ser visualizado da mesma maneira.

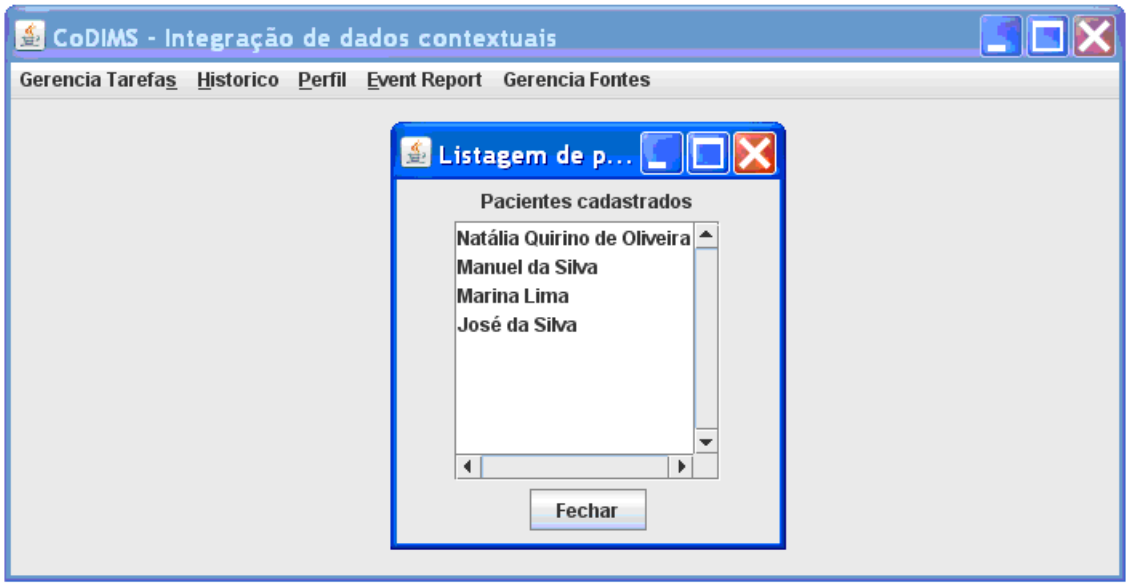

Figura 77. Listagem de dados contextuais

c) **Interface gráfica listagem de eventos:** todas as ocorrências em seus detalhes. Futuramente, poderão ser filtradas por diversos parâmetros, tais como tipos e faixa de horário nos quais ocorreram.

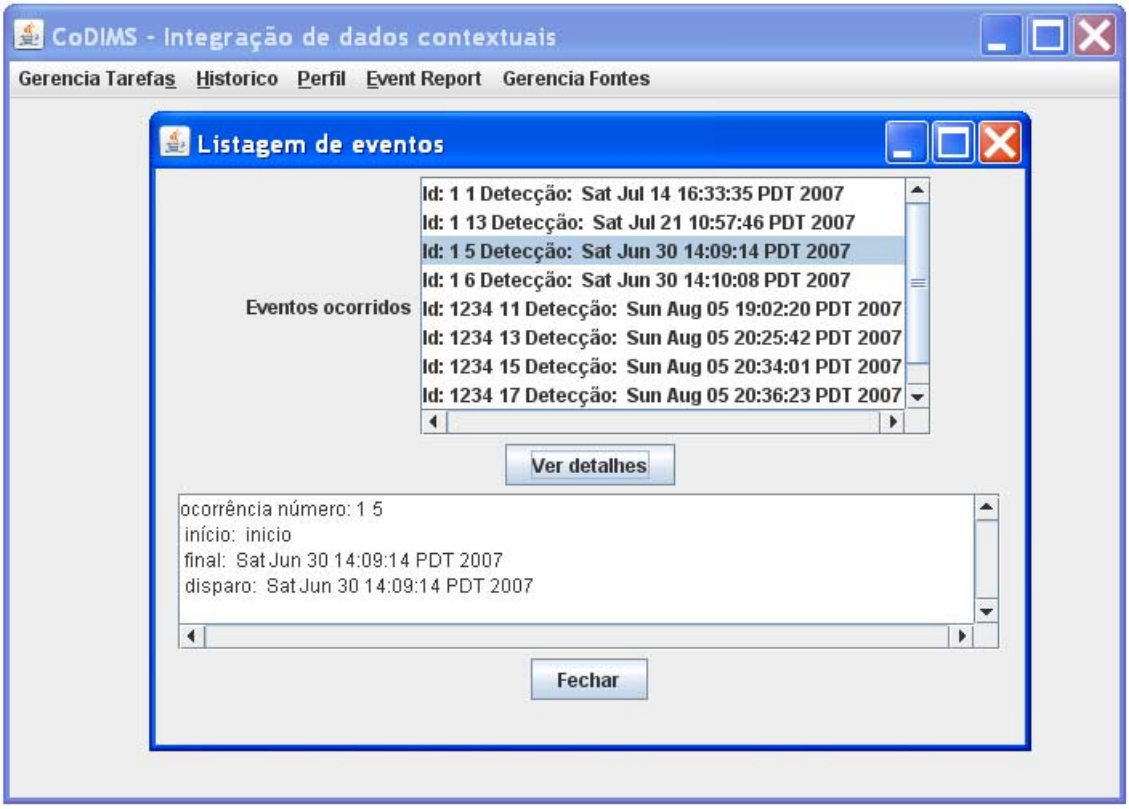

Figura 78. Listagem de eventos ocorridos

A Figura 79 mostra uma lista sugerida para operadores disponíveis.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.w3schools.com"
  xmlns="http://www.w3schools.com">

Operadores unarios:
<xs:element name="NOT" minOccurs="0" maxOccurs="unbounded">
<xs:element name="amentar" minOccurs="0" maxOccurs="unbounded">
<xs:element name="diminuir" minOccurs="0" maxOccurs="unbounded">
<xs:element name="maiorque" minOccurs="0" maxOccurs="unbounded">
<xs:element name="igual" minOccurs="0" maxOccurs="unbounded">
<xs:element name="soma" minOccurs="0" maxOccurs="unbounded">
<xs:element name="contador" minOccurs="0" maxOccurs="unbounded">

Operadores de repetição
<xs:element name="repetir" minOccurs="0" maxOccurs="unbounded">
   <xs:attribute name="numeroVezes" type="xs:string"/>
   <xs:attribute name="lifetime" type="xs:string"/>
</xs:element>

operadores binários para eventos compostos:
   <xs:element name="conector">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="AND"/>
      <xs:enumeration value="OR"/>
      <xs:enumeration value="AdepoisDeB"/>
      <xs:enumeration value="AeBimultaneos"/>
     </xs:restriction>
    </xs:simpleType>
 </xs:element>
<operadores sobre ações em uma fonte de dados>
<xs:element name="aoObterDados" minOccurs="0" maxOccurs="unbounded">
<xs:element name="aoAtualizar" minOccurs="0" maxOccurs="unbounded">
<xs:element name="aoDeletar" minOccurs="0" maxOccurs="unbounded">
</xs:schema>
```

Figura 79. Lista de operadores

d) **Interface gráfica para listagem de subscrições:** nesta tela, o administrador ou usuário final pode visualizar as subscrições criadas, e informações como o plano de execução, os agentes criados para esta subscrição, e o estado atual da execução.

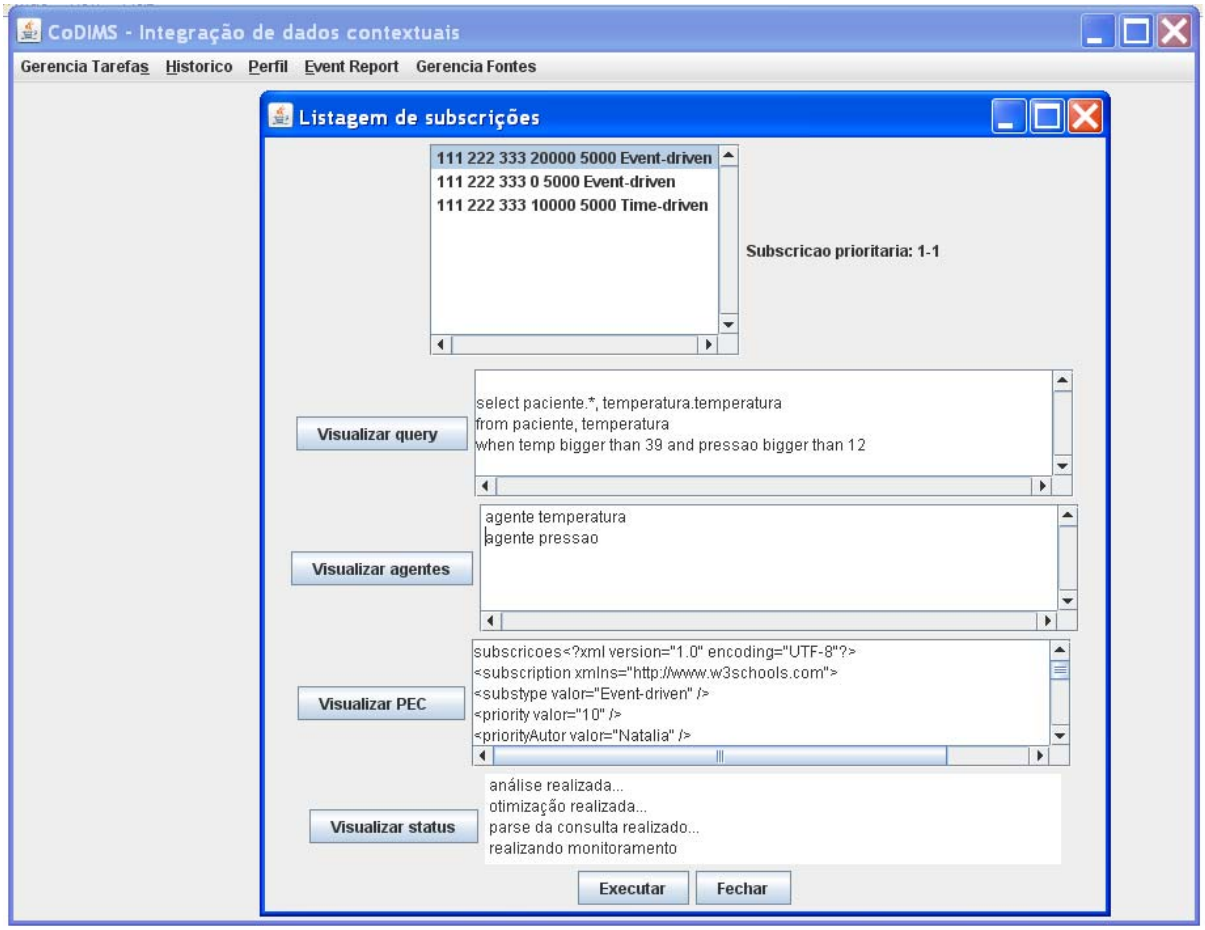

Figura 80. Listagem de subscrições

Através do monitor de sinais vitais, visto na Figura 81, é possível ver de uma maneira intuitiva os valores do eletrocardiograma.

Figura 81: Monitor de sinais vitais (extraído de [GONÇALVES, 2006])

e) **Interface gráfica para a definição pelo administrador de prioridade a uma subscrição:** apresentada na Figura 82. O nível de prioridade escolhido definirá a seqüência de criação de MECs e execução de consultas.

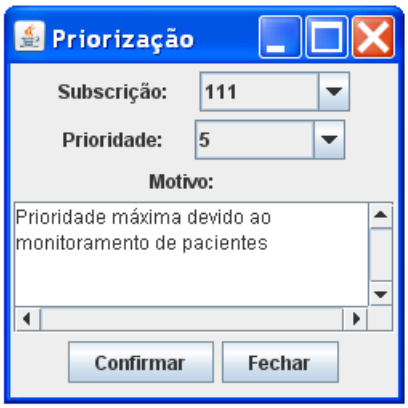

Figura 82. Configuração de prioridade de evento

f) **Interface para modificação de um perfil de usuário:** podem ser visualizados todos os perfis de usuário criados, com seus valores e demais informações. Futuramente podem ser feitas melhorias, tais como filtragem por valores e obrigatoriedade.

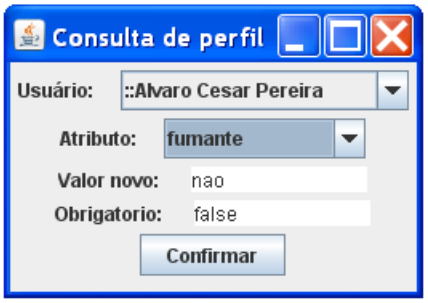

Figura 83. Interface de modificação de perfil

Para demonstrar o funcionamento de perfis de usuário, suponha a seguinte consulta referente à busca por médicos de uma especialidade:

```
SELECT * FROM medico m
WHERE m.especialidade = “oftamologista”
```

Figura 84: Consulta para perfil de usuário

O perfil a seguir representa a preferência de um usuário por médicos localizados em seu bairro, mas não obrigatoriamente, e onde o médico deve estar conveniado ao seu plano de saúde. Este plano deve ser modificado de acordo com este perfil de usuário, para adequação ao contexto do usuário, sem que o usuário precise mencioná-lo explicitamente.

```
<perfil>
<coluna nome="bairro" valor="Jardim da Penha" eliminatorio="false" />
<coluna nome="convenio" valor="Casufes" eliminatorio="true" />
</perfil>
```

Figura 85: Perfil de usuário

A consulta personalizada é então gerada a partir destes dois arquivos. Um exemplo é visto a seguir:

```
SELECT * from medico m
WHERE m.especialidade = “oftamologista” AND m.convenio = “casufes”
ORDER BY m.bairro
```

Figura 86: Consulta adequada a perfil

Desta forma, algumas condições são eliminatórias, outras são simplesmente classificatórias (identificadas pelos operadores “*order-by*”).

g) **Interface para monitoramento de dados históricos:** nesta tela é possível configurar quais os dados de quais atributos serão armazenados, e qual a freqüência, além de se visualizar os dados existentes.

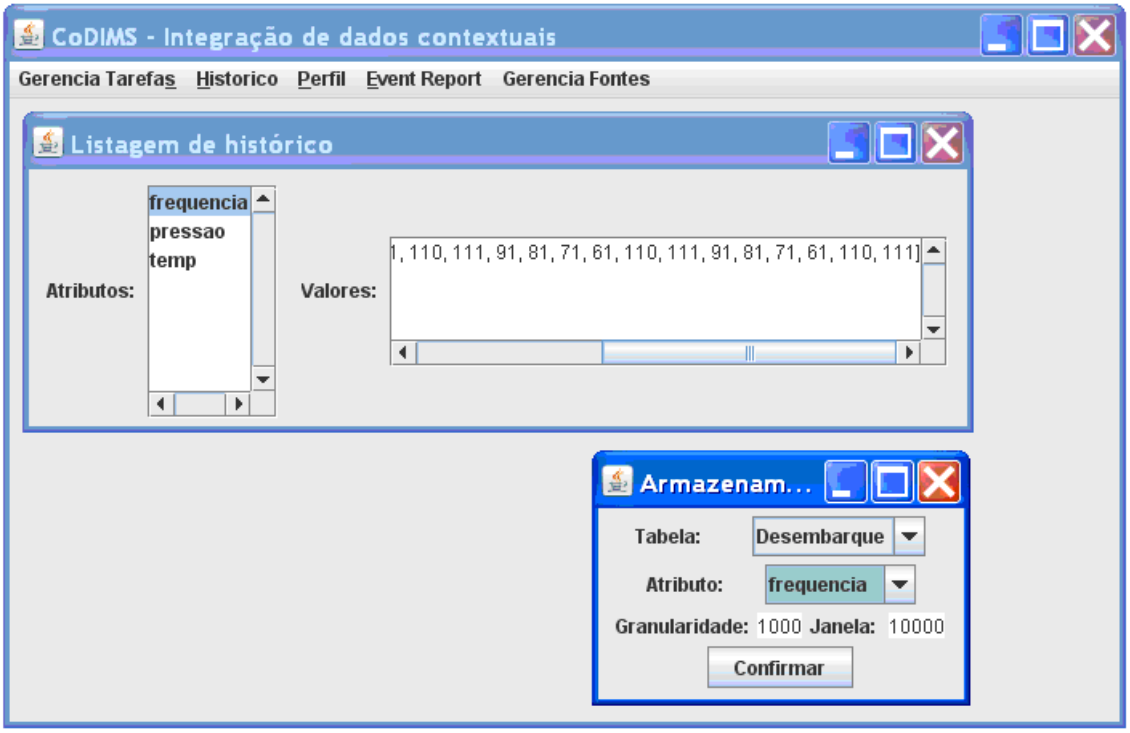

Figura 87. Configuração e obtenção de dados históricos

Os dados históricos, como os visualizados na figura anterior, são obtidos de arquivos XML, que são incrementados à medida que os sensores obtêm novos dados. Vemos na Figura 88 o arquivo XML para os valores da Figura 83. Pode-se ver que são descritos o horário de início do armazenamento, assim como a freqüência de atualização (no caso, a cada minuto).

```
<histórico xmlns="http://www.w3schools.com">
<atributo>ECG</atributo>
<tabela>sinaisVitais</tabela>
<inicioArmazenamento>10:02:05</inicioArmazenamento>
<granularidade>00:01:00</granularidade>
<valores>
   <valor>110</valor><valor>111</valor><valor>91</valor><valor>81</valor>
   <valor>71</valor><valor>61</valor>...
</valores></historico>
```

Figura 88: Atualização do histórico

h) **Interface para Gerência de fontes:** O gerenciamento da disponibilidade das fontes é exibido na Figura 89 com os períodos de indisponibilidade.

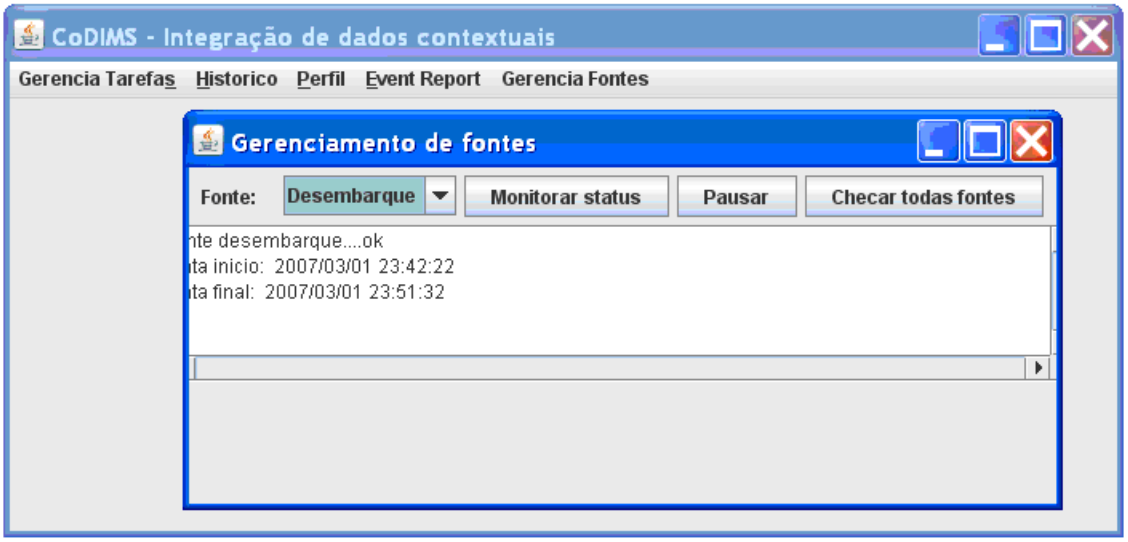

Figura 89: Interface de gerenciamento de fontes

Na Figura 90, um exemplo de descrição XML de um evento ocorrido. O conjunto mínimo de dados são os horários de todos os sub-eventos correspondentes e seu *status* atual. Vale lembrar que eventos parcialmente atingidos podem ter semântica particular e útil em ambientes sensíveis ao contexto.

```
<eventReport><tipo>anomaliaECG</tipo>
<idSubscricao>1234</idSubscricao>
<idEvento>5678</idEvento>
<timestampOcorrencia>2007-05-02 10:21:59</timestampOcorrencia>
<timestampDisparo>2007-05-02 10:22:01</timestampDisparo>
</eventReport>
```

Figura 90: Registro de evento

6.7 Gerenciamento de subscrições

Um levantamento inicial foi realizado para descobrir qual a melhor maneira de modelar eventos e ações contextuais. Primeiramente, são armazenadas informações como o *id* da subscrição, *id* da aplicação requisitante, *ids* de usuário e grupo, tipo e classe, prioridade e *timeout* (ou seja, hora no qual a consulta deverá ser encerrada). Também é indicado para quem os resultados devem ser retornados (módulo de subscrição, de privacidade ou interpretador de contexto). Finalmente, os eventos a serem monitorados são descritos.

Pode haver várias subscrições simultâneas, constituindo um sistema multi-usuário. Para isto torna-se necessário o controle de prioridade, que é realizado pelo Gerente de Tarefas, sob responsabilidade do administrador. Um exemplo de fila de entrada é visto a seguir:

Tabela 23: subscrições concorrentes

Subscrição	***Aplicação requisitante***	***Prioridade***
Subscrição 03	Interpretador de contexto 01	1.0 - Tempo real
Subscrição 01	Módulo de privacidade	0.8 - Alta
Subscrição 02	Interpretador de contexto 02	0.55 - Média

Segundo [POEL, 2002], os estados possíveis de uma subscrição em sistemas sensíveis ao contexto são os estados mostrados na Figura 91:

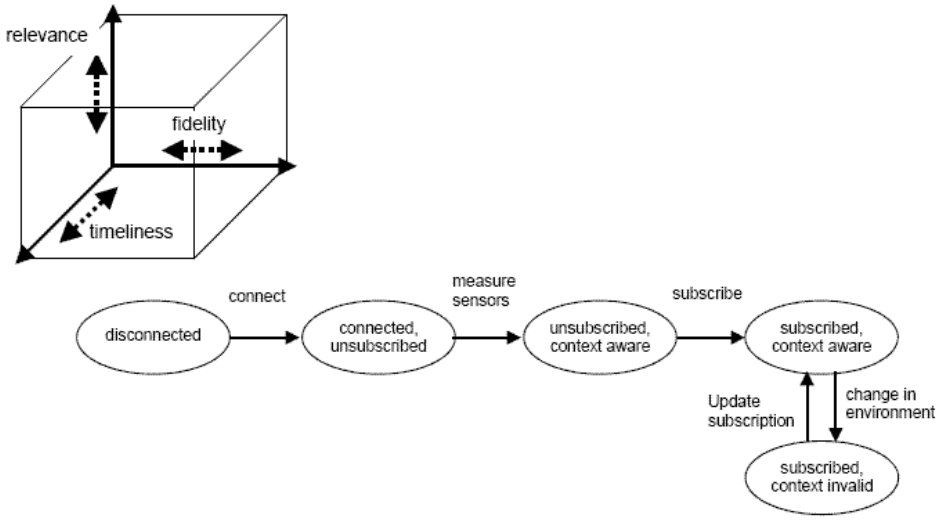

Figura 91. Diagrama de estados para subscrições.

6.8 Ambiente de Implementação e Ferramentas Utilizadas

As ferramentas utilizadas foram: Eclipse; servidor Tomcat e SOAP para invocação de serviços *web*; parser JDOM; *driver* JDBC para PostgreSQL. Tais escolhas refletem as seguintes preocupações:

- **Ferramentas gratuitas ou livres:** Sendo o DBMWare e o Telecardio iniciativas de pesquisa com apoio governamental, deve-se dar prioridade a ferramentas gratuitas.
- **Padrões:** ir sempre que possível de acordo com padrões emergentes ou em consolidação na comunidade de ciência da computação.

6.8.1 SAX: Biblioteca para programação XML

Para acessar a um documento XML, pode-se usar a linguagem de programação que se desejar. Uma linguagem típica para trabalhar com XML é Java e, neste caso é a SUN *Microsystems* a

encarregada de prover os mecanismos necessários para acessar os documentos XML e trabalhar com eles. Um desses mecanismos é o SAX.

O *Simple API for XML* [SAX, 2007] é um conjunto de interfaces (APIs) que devem ser implementadas para efetuar o *parsing* de arquivos XML. O SAX é uma API baseada em eventos, o que significa que as funções (ou métodos, dependendo da linguagem de programação usada) são chamadas pelo analisador SAX. Será, por exemplo, chamada uma função no início e no fim do *parsing* de cada elemento.

Há dois principais tipos de APIs para XML:

- *Tree-based*: mapeia um documento XML em uma estrutura interna tipo árvore, e então, permite que uma aplicação navegue por ela. O grupo de trabalho *Document Object Model* (DOM) do W3C mantém a API para documentos XML e HTML baseada em árvore;
- *Event-based*: uma API baseada em evento informa a análise de eventos (tal como o começo e o fim dos elementos) diretamente para a aplicação através de *callbacks*, e não constroem uma árvore interna. A aplicação implementa um manipulador (*handlers*) para tratar com diferentes eventos. Uma vantagem desta abordagem é que se torna desnecessário construir uma árvore representando o documento XML, que pode ser grande, poupando-se memória [CÔCO, 2005]. SAX é o melhor exemplo de tal API.

O SAX inicia seu processamento lendo o elemento raiz do documento, e depois, realizando um percurso na seqüência dos elementos do documento XML.

6.8.2 JDOM

O JDOM é uma biblioteca para se manipular documentos XML utilizado neste trabalho para a realização dos parsers de subscrições e para a implementação do padrão de projeto *Direct Access Object* (DAO). Um exemplo de leitura de arquivo é mostrado abaixo, no caso, para leitura de um *event report*:

```
try{EventReport e = new EventReport();
    File f = new File(nomeArq);
    SAXBuilder sb = new SAXBuilder();
    Document d = sb.build(f);
    Element raiz = d.getRootElement();
    Namespace ns =  Namespace.getNamespace("http://www.w3schools.com");
    System.out.println(raiz.getChildren().size());
    Element evInicial = (Element) raiz.getChild("eventoInicial",ns);
    e.setTimestampInicio(evInicial.getAttributeValue("timestamp"));
catch(Exception e)
    {e.printStaceTrack();}
```

Figura 92: Execução da biblioteca JDOM

7 Conclusões, contribuições e trabalhos futuros

Os grandes avanços nas tecnologias de computação e comunicação móvel propiciaram a popularização de dispositivos tais como celulares e PDAs e a maior presença da computação nas atividades do dia-a-dia. Neste cenário, as aplicações sensíveis ao contexto potencializam estes avanços ao aprimorar a interação com os seus usuários através da obtenção automática de informações sobre o contexto físico, temporal e computacional do ambiente que os cerca. Porém, as infra-estruturas existentes para o desenvolvimento de aplicações sensíveis ao contexto, geralmente tratam superficialmente o problema da integração de dados e foram projetadas para domínios específicos. Essas limitações geraram a motivação para o desenvolvimento deste trabalho. A arquitetura proposta é flexível e configurável, atendendo assim o dinamismo de tais aplicações, além de poder ser adaptada a diferentes domínios.

A arquitetura proposta é uma nova instância do CoDIMS, que em sua proposta inicial não contempla a entrega de informações de forma ativa: as requisições são realizadas pelos usuários tal como nos SGBDs tradicionais (*request-response*). Sendo assim, foram necessárias dentre outras: alterações em todo o pocessamento de consultas; o desenvolvimento de novos *wrappers;* o desenvolvimento de novos componentes; incorporar uma fábrica de MECs e uma fábrica de agentes.

7.1 Comparações com Trabalhos Relacionados

No capítulo 4 foram realizadas comparações com os trabalhos relacionados, e levantados o conjunto de requisitos apontados por cada um deles, que variam devido às suas origens e propósitos. Foi mostrado que o projeto Awareness é o mais semelhante à proposta aqui apresentada, porém com importantes distinções, em particular na entrega de dados. Este trabalho distribui a execução de consultas e a detecção de eventos, o que traz benefícios às plataformas sensíveis ao contexto.

Na arquitetura proposta, todas as informações referentes à conexão com as fontes se encontram distribuídas, o que permite a continuação das consultas já realizadas mesmo que o componente Metadados falhe. O CoDIMS-CA permite a adição e substituição de novos componentes para os diversos estágios de processamento, e para a representação das fontes, consultas, eventos, estatísticas de acesso e resultados da consulta, de acordo com os interesses

de cada aplicação. Alguns trabalhos relacionados possuem funcionalidades extras, mas que foram julgadas necessárias no escopo do Infraware: por exemplo, o projeto Cream mesclando eventos vindos de vários sistemas.

Este trabalho utiliza ontologias para determinar a correspondência entre os perfis de usuário definidos semanticamente, e os atributos disponibilizados no banco de dados distribuídos. Ou seja, as preferências de usuário são embutidas nas consultas, eliminando acessos a fontes no caso de haver fragmentação horizontal.

Durante a implementação do CoDIMS-CA para o cenário descrito no capítulo 6, torna-se possível validar a arquitetura proposta e a comparação com os trabalhos relacionados.

- Mogatu: esta arquitetura não permite a priorização das consultas referentes à frequência cardíaca, tampouco histórico contextual e facilidades de *event report*.
- Nexus: falta de adaptação para o domínio da telemedicina
- Awareness: caso a mesma aplicação fosse modelada utilizando-se a arquitetura, haveria um maior tráfego de *resultsets* entre as unidades base e a central.

Além disto, este cenário demonstra que todas as funcionalidades demonstradas no capítulo 5 podem ser necessárias simultaneamente.

Finalmente, a Tabela 24, sintetiza as diferenças entre as abordagens comparadas, e mostra o diferencial do CoDIMS-CA em relação a elas.

Tabela 24: Comparação de requisitos para plataformas sensíveis ao contexto.

	CoDIMS-CA	**Awareness**	**Nexus**	**MoCA**	**DBGlobe**	**Mogatu**
Extensibilidade	**X**	**X**	**X**	**X**		**X**
Heterogeneidade	**X**	**X**		**X**	**X**	**X**
Metadados	**X**	**X**	**X**	**X**	**X**	**X**
Dinamismo	**X**		**X**	**X**	**X**	**X**
Regras ACID			**N/E**			
Perfil de usuário	**X**	**X**	**N/E**	**X**	**X**	**X**
Entrega ativa	**XX**	**X**	**X**	**X**	**X**	**X**
Prioridade	**X**		**N/E**			
Histórico	**X**	**X**	**N/E**			
Inicialização dinâmica	**X**	**N/E**	**X**	**X**	**N/E**	**X**
Mobilidade			**X**	**X**	**N/E**	**X**
espaço-temporal			**X**			**X**
Linguagem eventos	**X**	**X**	**X**	**X**	**X**	**X**
Linguagem espacial			**X**			

Portanto, a partir da tabela 24 justifica-se a implementação do CoDIMS-CA como opção flexível para vários domínios de aplicações sensíveis ao contexto.

7.2 Requisitos Atendidos

A Tabela 25 mostra os requisitos já atendidos pelo CoDIMS-CA, e os requisitos postergados, constituindo trabalhos futuros.

Requisito	Situação	Observação
Acesso e integração de dados contextuais	Atendido	
Dinamismo das informações	Parcialmente atendido	A Gerência de Fontes informa a disponibilidade das fontes, mas o otimizador ainda não atribui pesos dstintos às fontes de acordo com sua conectividade.
Metadados	Atendido	
Perfil de usuário	Atendido	
Contexto espaço-tempo-computacional	Não atendido	O uso de GPS é indicado como um trabalho futuro, e o LPRM já adquiriu dispositivos que podem ser utilizados para tal tarefa
Configuração e escalabilidade	Atendido	
Entrega ativa	Atendido	
Modificações na ACID	Não atendido	Este tópico é por si só uma grande área de pesquisa em Banco de Dados.
Histórico contextual	Atendido	
Mobilidade	Parcialmente atendido	Este tópico é por si só uma grande área de pesquisa em Banco de Dados. Em nosso estudo de caso, a funcionalidade da base móvel e do Holter foram garantidos pela Engenharia Elétrica e pelo criador do WrapperECG. Como os wrappers são responsáveis pelo acesso às fontes de dados, estes são trbalhos futuros dependentes
Adição dinâmica de fontes de dados	Parcialmente atendido	O usuário deve criar o *wrapper* para a fonte e disponibilizar através dos metadados, mas ela é adicionada sem necessidade de interromper o funcionamento do CoDIMS.
Prioridade entre consultas	Atendido	
Linguagem de eventos	Atendido	
Linguagem espaço-temporal	Não atendido	As funcionalidades de criação de operadores no CoDIMS podem auxiliar neste trabalho futuro

Tabela 25. Atendimento aos requisitos.

7.3 Contribuições

Nossa abordagem é uma máquina de execução de consultas distribuída e baseada em componentes, que possa lidar com a detecção distribuída de eventos em bases de dados

distribuída, e executar consultas periodicamente. Para isto, existe a necessidade de fábricas de agentes, e distribuição automática de agentes e operadores entre os nós que contém as fontes de dados. Benefícios adicionais são a reusabilidade e manutenabilidade, devido ao desenvolvimento baseado em componentes.

A contribuição deste trabalho, além da arquitetura proposta, também apresenta uma lista de requisitos a serem atendidos por sistemas de acesso e integração de dados para aplicações sensíveis ao contexto. Pretende-se que futuramente esta arquitetura seja utilizada em diferentes domínios de aplicação, o que justifica a extensibilidade e reconfiguração como objetivos principais, indo ao encontro às principais características do CoDIMS.

Uma das justificativas para a integração de dados é a verificação de sua necessidade para bases de dados legadas que se encontram em operação por anos. A existência de dados de formatos proprietários também torna obrigatória esta funcionalidade em detrimento de um projeto centralizado.

Para validar a arquitetura foram implementadas versões iniciais de seus componentes e um estudo de caso com uma aplicação em tele-medicina no projeto Telecardio. A flexibilidade, extensibilidade e configuração desejada foram obtidas com o auxílio de tecnologias como *Web* S*ervices* e XML, e de metodologias de desenvolvimento baseado em componentes e *frameworks*. Essas características têm auxiliado significativamente a incorporação dos requisitos apresentados na seção 2.

Mostraram-se neste trabalho os fluxos de dados mais relevantes, além de decisões de projeto cruciais na arquitetura. O módulo se encontra atualmente em estágio de implementação. As contribuições alcançadas foram:

- Tornou-se claro quais componentes do CoDIMS devem ser modificados e quais novos módulos deveriam ser criados para se alcançar a funcionalidade desejada, com flexibilidade para mudanças futuras;
- Demonstrou-se a eficácia do *framework* CoDIMS através de sua reutilização;
- Especificação UML dos novos componentes, classes, transições de estado e fluxos de dados a serem desenvolvidos, baseados nos requisitos e casos de uso identificados;
- Todas as classes definidas na Figura 33 foram implementadas. Dos requisitos mencionados no Capítulo 3, apenas o aspecto espacial, o tratamento de perfis e a validade temporal não foram estudados;

- No momento, novos exemplos de aplicações estão sendo desenvolvidos, principalmente para validar a arquitetura em diferentes condições;
- Definição de vários formatos, entre eles o de perfil, armazenamento e consulta de histórico contextual, planos locais e conjunto-resultados;
- Estão sendo realizados testes de integração com as demais camadas do projeto Telecardio, em particular a camada de interpretação de contexto, com o envio de novas subscrições a serem executadas;
- Foram submetidos e aceitos artigos para congressos como o Simpósio Brasileiro de Redes de Computadores (SBRC2006) [PEREIRA FILHO *et.al.*, 2006] e *Workshop on Ongoing Thesis and Dissertations* (em conjunto com o Webmedia 2007) [OLIVEIRA; BARBOSA, 2007].

7.4 Trabalhos Futuros

Algumas questões não foram exploradas no escopo deste trabalho, constituindo trabalhos a serem desenvolvidos, dentre eles:

- **Controle de acesso e privacidade:** A criação de visões não foi tratada, sendo realizada pelo Módulo de Privacidade da plataforma Infraware. Esta funcionalidade é crucial para prover segurança em um ambiente de integração de dados com várias organizações e atores presentes. O controle de acesso se reflete no disparo de notificações apenas para pessoal autorizado;
- **Estudo do tráfego em rede** e mensagens trocadas entre os *web services* está sendo realizado por membros do projeto Infraware. Este estudo é importante para se validar e aperfeiçoar os fluxos para grandes volumes de dados. Além disto, auxilia a atender o requisito de dinamismo nas informações, visto que eles não foram abordados suficientemente neste trabalho;
- **Incorporação do uso de *grids* computacionais** para processamento de consultas distribuídas fornecendo escalabilidade e desempenho quando houver grande número de fontes extensas de dados;
- **Desenvolvimento de novos *wrappers:*** para diferentes domínios e diferentes formatos de dados; *wrappers* para diversos dados médicos, como imagens a serem visualizadas em um ambiente de *grid* já são vislumbrados no contexto do sistema de saúde do

Espírito Santo; *wrappers* GPS auxiliariam no atendimento a consultas *location-aware* e o requisito de contexto espacial;

- **Linguagens para contexto espacial:** a partir do desenvolvimento de *wrappers para localização*, como por exemplo, GPS, pode-se definir uma linguagem para a representação espacial de objetos;
- **Otimização dinâmica de consultas:** em tempo de execução e usando dados contextuais. Esta otimização pode se valer de informações sobre a disponibilidade das fontes no momento e do perfil do usuário, como por exemplo, idioma, agenda de compromissos e localização;
- **Estudo do relaxamento das regras ACID:** grande esforço tem sido dedicado ao estudo de transações de BD em ambientes desfavoráveis;
- **Tolerância e tratamento de falhas:** quando um componente estiver fora de serviço, deve-se procurar outro disponível para substituí-lo. No momento tal funcionalidade não está prevista, mas pode ser incorporada no componente Controle;
- **Linguagem de consulta para eventos:** onde se possam buscar as emergências ocorridas, filtradas por horário, tipo de evento, horários com maior ocorrência;
- **Ações administrativas:** estas ações, realizadas pelo administrador, são ações de suporte da plataforma, como para criar um novo usuário fornecer acesso de administração. Até o presente momento, não foram implementadas;
- **Estatísticas:** entre as novas estatísticas armazenadas estão as fontes com maior disponibilidade e confiabilidade, para quando for necessário evitar fontes intermitentes;
- **Aperfeiçoamentos no Gerente de Fontes:** melhorias podem ser realizadas a fim de que sejam indicados em detalhes os motivos da indisponibilidade de fontes, uma maior integração com as estatísticas citadas anteriormente;
- **Aperfeiçoamento no componente Perfil:** sendo este trabalho um estudo em amplitude, não foi possível desenvolver em amplitude todos os componentes necessários. Algoritmos mais complexos para a reescrita com perfis podem ser criados.

Referências Bibliográficas

ABITEBOUL, S. et. al. (2005) The Lowell Database Research Self Assessment. Communications of the ACM, Vol. 48 No. 5, 111-118.

ANDREAO, R.V., FILHO, J.G.P., CALVI, C.Z. (2006) TeleCardio: Telecardiologia a serviço de pacientes hospitalizados em domicílio. Congresso Brasileiro de Informática em Saúde, 2006.

AYRES, F. V. M., PORTO, F. A. M., MELO, R. N. (2003) Uma máquina extensível para suporte a novos modelos de execução de consultas. Simpósio Brasileiro de Banco de Dados, 2003.

AYRES, F.V.M. (2003) QEEF: uma máquina de execução de consultas extensível. Tese de doutorado, PUC-Rio, Rio de Janeiro, 2003.

BARBOSA, A.C.P. (2001) Middleware para Integração de Dados Heterogêneos Baseado em Composição de Frameworks Tese de Doutorado, PUC-Rio, Rio de Janeiro, 2001.

BARBOSA, A.C.P., PORTO, F., MELO, R.N. (2002) Configurable Data Integration Middleware System. Journal of the Brasilian Computer Society. Vol. 8 No. 2, 12-19

BAUER, M. et. al. (2004) Information Management and Exchange in the Nexus platform. Relatório técnico 2004/04, Stuttgart, Germany, 2004. Disponível em ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/TR-2004-04/TR-2004-04.pdf

BECHHOFER, S., CARR, L., GLOBE, C., KAMPA, S., MILES-BOARD, T. (2002) The semantics of semantic annotation. ODBASE: First International Conference on Ontologies, Databases, and Applications of Semantics for Large Scale Information Systems, Irvine, 2002.

BEMMEL, J.V, DOCKHORN, P., WIDYA, I. (2004) Paradigm: Event-driven computing. Relatório técnico D2.7a. Disponível em https://doc.freeband.nl/dsweb/Get/Document-55291/

BERNAUER, M., KAPPEL, G., KRAMLER, G. (2004) Composite Events for XML. Proceedings of the 13th international Conference on the World Wide Web, NY, USA, 2004

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. (2001) The Semantic Web. Em: Scientific American (edição 50), maio de 2001. Disponível em: <http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21>

BIANCARDI, C., SILVESTRE, L. J., BARBOSA, A., C., P. (2005) A CoDIMS based proposal for distribution and execution of wrappers in a grid environment. Proceedings of the III Workshop on Computational Grids and Applications (WCGA 2005), LNCC, Petrópolis, RJ. Disponível em < codims.lprm.inf.ufes.br >

BIERMAN, G., BUNEMAN, P., GARDNER, P. (2003) Ubiquitous Data. UK-UbiNet Workshop, 2003. Disponível em < http://research.microsoft.com/~gmb/Papers/ubinet03.pdf >

BONNET, P., GEHRKE, J. E., SESHADRI, P. (2000) Querying the Physical World. IEEE Personal Communications, Vol. 7, No. 5, 10-15. Special Issue on Smart Spaces and Environments.

BRAY, T. (2001) What is RDF? Disponível em: <http://www.xml.com/pub/a/2001/01/24/rdf.html?page=1 >

BRAYNER, A., FILHO, J. A. M. (2002) AMDB: An Approach for Sharing Mobile Databases in Dynamically Configurable Environments. Simpósio Brasileiro de Banco de Dados, Gramado, 2002.

BUNNINGEN, A.H., APERS, P. (2005) Context for Ubiquitous Data Management. Centre for Telematics and Information Technology, Database Group, University of Twente, Netherlands, 2005. Disponível em < eeexplore.ieee.org/iel5/10199/32542/01521233.pdf?arnumber=1521233 >

CELENTANO, A., SCHREIBER, F., TANCA, L. (2004) Requirements for Context-Dependent Mobile Access to Information Services. Disponível em < http://www.dsi.unive.it/~auce/docs/celentano_mis04.pdf >

CHAKRAVARTHY, V., Krishnaprasad, E., Anwar, Kim, S.K. (1994) Composite Events for Active Databases: Semantics, Contexts and Detection. Proc. International Conference on Very Large Data Bases VLDB, Santiago, Chile, 1994

CHEN, H. (2004) An intelligent broker architecture for pervasive context-aware systems. Tese de doutorado, Georgia Institute of Technology.

CHEN, L., SHADBOLT, N. R., TAO, F., GLOBE, C., COX, S.J. (2003) Exploiting semantics for e-Science on the semantic grid. Web Intelligence (WI2003) workshop on Knowledge Grid and Grid Intelligence, 13-16 October 2003, Halifax, Canada Disponível em <www.geodise.org/files/Papers/KGGI_118_clm.pdf>

CHEN, L., SHADBOLT, N.R., TAO, F., GLOBE, C., PULESTON, C., COX, S.J (2004) Managing semantic metadata for the semantic grid. Disponível em < www.geodise.org/files/Papers/KGGI2004.pdf >

CILIA, M. (2002) An Active Functionality Service for Open Distributed Heterogeneous Environment. Tese de doutorado. 2002. Disponível em < www.dvs1.informatik.tu-darmstadt.de/publications/pdf/active-functionality.pdf >

CILIA, M., BORNHOVD, C., BUCHMANN (2003) CREAM: An infra-estructure for distributed, heterogeneous event-based applications. Conference on Cooperative Information Systems, vol. 2888 of LNCS, 482-502, Catania, Italy, November 2003.

CIUFFO, L. N. (2002) Linguagens e ferramentas para a Web Semântica. Monografia de graduação, Universidade Federal de Juiz de Fora, Juiz de Fora, 2002.

CÔCO, T.M. (2005) Implementando Wrappers XML e Relacional para o Codims. Monografia de graduação, Universidade Federal do Espírito Santo, Vitória, 2005.

CODIMS (2006) Disponível em <http://www.codims.lprm.inf.ufes.br>

COSTA, P. D. (2003) Towards a Services Platform for Context-Aware Applications. Dissertação de Mestrado, University of Twente, Enschede, The Netherlands. Disponível em < www.cs.utwente.nl/~dockhorn/files/thesis_dockhorn.pdf >

DA SILVA, V. F. V., DUTRA, M. L., PORTO, F., SCHULZE, B, BARBOSA, A.C.P., OLIVEIRA, J. C. (2006) An adaptive parallel query processing middleware for the Grid. Concurrency and Computation: Practice and Experience, Vol. 18, No. 6, 621-634.

DAYAL, U., HANSON, E., WIDOW, J. (1994) Active Database Systems.

DBMWARE (2005) Metamodelo de contexto e perfil UML-CW. Documento de Projeto, DBM-Ware/CNPq/D1.4, v.2.2

DE ROURE (2001) The Semantic Grid: A Future e-Science Infrastructure Disponível em < http://www.semanticgrid.org/documents/semgrid-journal/semgrid-journal.pdf >

DEY, A.K. (2000) Providing Architectural Support for Building Context-Aware Applications. Tese de Doutorado, Georgia Institue of Technology, 2000.

DEY, A.K (2001) Understanding and Using Context. Georgia Institute of Technology, Atlanta,USA, 2001. Disponível em < www.cc.gatech.edu/fce/ctk/pubs/PeTe5-1.pdf >

EKKEBUS, S.P. (2006) Telemonitoring services for home care ventilators, using web services. Dissertação de mestrado, University of Twente, Twente, 2006

ERFIANTO, B. (2004) Design of a vital sign protocol format using XML and ASN.1. Dissertação de mestrado, University of Twente, Twente, 2004.

FAYAD, M., SCHMIDT, D., JOHNSON, R. (1999) Building application frameworks object-oriented foundations of frameworks. John Wiley & Sons Inc., 1999

FDA (2007) U.S. Food and Drug Administration. Disponível em <www.fda.gov/>

FISCHER, J.G. (2004) Spatial Context Management for Spatial Context Management for Augmented Reality in Vehicles, dissertação de mestrado.

FONTOURA, M., PREE, W., RUMPE, B. (2000). UML-F: A Modeling Language for Object-Oriented Frameworks. 14th European Conference on Object Oriented Programming (ECOOP 2000).Disponível www.almaden.ibm.com/cs/people/fontoura/papers/ecoop2000.pdf

FURUIE, R. *et. al*. (2003) Prontuário Eletrônico de Pacientes: integrando informações Clínicas e imagens médicas. Revista Brasileira de Engenharia Biomédica.Vol.19, 125-137. São Paulo.

GAMMA, E.; RICHARD H.; RALPH J.; JOHN V. (1995) Design Patterns: Elements of Reusable Object-Oriented Software. Editora Addison-Wesley. ISBN 0201633612, 1995.

GATZIU, S., DITTRICH, K.R. (1994) Detecting composite events in active database systems using Petri Nets. Proceedings of the 4th International Workshop on Research Issues in Data Engineering, 1994

GERGATSOULIS, M., STRAVAKAS, Y. (2003) Representing Changes in XML Documents using Dimensions. Disponível em www.springerlink.com/index/J5B44TGVUWXD0NL5.pdf

GLOBE, C. High level Knowledge-based Grid Services for Bioinformaticans, 2003 Disponível em: <http://twiki.mygrid.org.uk/twiki/bin/view/Mygrid/AccessGrid01May2003.ppt>

GML (2006) <http://schemas.opengis.net/gml/3.1.1/base/ >

GOMES, F.R. (2005) Otimizador de Consultas do CoDIMS. Monografia de Graduação, Universidade Federal do Espírito Santo, Vitória, 2005. Disponível em http://codims.lprm.inf.ufes.br

GONÇALVES, B. N. (2006): Projeto de um ECG-Wrapper para a Plataforma Infraware. Projeto de Graduação. Departamento de Informática. Universidade Federal do Espírito Santo.

GRAEFE, G., DEWITT, D. J. (1987) The exodus optimizer generator. In SIGMOD Conf., pages 160–172, 1987

GRAY, P.D., SALBER.D. (2001) Modelling and using sensed context information in the design of interactive applications. In: Little, M. R. and Nigay, L. (eds). Engineering for Human-Computer Interaction. Lecture Notes in Computer Science (2254), Springer-Verlag, pp. 317-336

GRISWOLD, W.G., BOYER, R., BROWN, S.W., TRUONG,T.M. (2003) A Component Architecture for an Extensible, Highly Integrated Context-aware computing infrastructure. Proceedings of the 25th International Conference on Software Engineering , p.363-372

GRUBER, T. R.A (1993) Translation approach to portable ontologies. Knowledge Acquisition, 5(2):199-220, 1993. Disponível em <ftp://ftp.ksl.stanford.edu/pub/KSL_Reports/KSL-92-71.ps.gz >

GUMO (2006) General Use Modelling Ontology. Disponível em <www.gumo.org>

HALEVY, A. Y. (2003) Data integration: A status report. In Proceedings of 10th Conference on Database Systems for Business Technology and the Web (BTW 2003), Germany.

HAKIMPOUR, F., GEPPERT, A. (2001) A. Resolving semantic heterogeneity in schema integration: an ontology based approach. In: Proceedings of the international conference on Formal Ontology in Information Systems. USA: [s.n.], 2001. v. 2001, p. 297 308.

HECKMANN, D. (2002) Proposal for a user modeling markup language (UserML). Disponível em < w5.cs.uni-sb.de/publication/file/85/Heckmann02ABIS.ps >

HEER, J. *et al* (2003) Liquid: Context-aware distributed queries". In Proceedings of Fifth International Conference on Ubiquitous Computing: Ubicomp, 2003

HENRICKSEN, K., INDULSKA, J., RAKOTONIRAINY, A. (2002): Modeling Context Information in Pervasive Computing Systems. In Proceedings of the First International Conference on Pervasive Computing (Pervasive´02).

HENRICKSEN, K., INDULSKA, J., RAKOTONIRAINY, A. (2003) Generating Context Management Infrastructure from High-Level Context Models. Proceedings of the 4th International Conference on Mobile Data Management, 1-6, Melbourne, Australia, 2003.

HÖNLE, N., KÄPPELER, U.,NICKLAS, D., SCHWARZ,T.,GROSSMANN,M. (2005) Benefits of Integrating Meta Data into a Context Model. Pervasive Computing and Communications Workshops, 2005.

IFTIKHAR, N. LIAQUAT, H. QADIR, M.A. (2006) Profile based Context-Aware Query Processing Architecture. Multitopic Conference, 2006. Disponível em: <ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4196415>

INDULSKA, J., et al. (2003), Experiences in Using CC/PP in Context-Aware Systems. Proc. of the 4th International Conference on Mobile Data Management, 247-261, Melbourne, Australia, 2003. Disponível em <www.springerlink.com/index/7LUTBG87WYUQF3CK.pdf

JAEGER, U., OBAMAIER, J. K. (1997) Parallel event detection in active database systems: The heart of the matter. ARTDB-97 - The Second International Workshop on Active, Real-Time, and Temporal Database Systems, Milão, Itália, 1997

JDOM (2007) Disponível em <www.jdom.org/>

JEFFERY, K. (1999) Knowledge, Information and Data. CLRC Information Technology Department, 1999. Disponível em: <http://www.escience.clrc.ac.uk/Publications/9/KnowledgeInformationData.doc>

JUDD, STEENISTE (2003) Providing Contextual Information to Pervasive Computing Applications. Disponível em < www.cs.cmu.edu/~aura/docdir/Percom03.pdf >

JUDE (2006) Jude System Design Tool. Disponível em <http://jude.change-vision.com/ >

LAQUINE, C.E. (2006) Envio de código dos wrappers em ambiente de grid para o CoDIMS Monografia de graduação, Universidade Federal do Espírito Santo, Vitória, 2006.

LAM, k., KUO, T., TSANGm W., LAW, G.C.K. Law (2000) Concurrency Control in Mobile Distributed Real-time Database Systems. Information Systems, vol. 25, no. 4, June 2000, 261-322.

KAPSAMMER, E., SCHWINGER, W., RETSCHITZEGGER, W. (2005) Bridging relational databases to context-aware services. Workshop CaiSE - Advanced Information Systems Engineering, 17th International Conference, 2005, 703-716, Porto, Portugal.

KOUTRIKA, G., IOANNIDIS,Y. (2004) Personalized Queries Using a Generalized Preference Model. Hellenic Data Management Symposium (HDMS), Atenas, Grécia

MAN, K., LAW, G.C.K., LEE,V.C.S (2000) Priority and deadline assignment to triggered transactions in distributed real-time active databases

MAYRHOFER, R. (2004) AN ARCHITECTURE FOR CONTEXT PREDICTION. Tese de doutorado. Disponível em < www.mayrhofer.eu.org/downloads/publications/PhD-ContextPrediction-2004.pdf >

MANTORO, T., JOHNSON, C. W. (2003) Location History in a Low-cost Context Awareness Environment, Workshop on Wearable, Invisible, Context-Aware, Ambient, Pervasive and Ubiquitous Computing, Australian Computer Science Communications, Vol. 25, No. 6, Adelaide, Australia, February 2003

MANTOVANELI, R.P. (2006) Dissertação de mestrado, Universidade Federal do Espírito Santo, 2006.

MCFADDEN, T. HENRICKSEN K., INDULSKA, J. (2004) Automating Context-aware Application Development.

MDA (2006) Model-driven Architecture http://www.omg.org/mda

MEIER, R., CAHILL, V. (2005) Taxonomy of distributed event-based programming systems. The Computer Journal, v48, No 5, 602-626.

MENA E., ILLARRAMENDI A., KASHYAP V., SHETH A. P. (2000) Observer: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. Distributed and Parallel Databases, 8(2):223–271, 2000.

MENEQUINI, R.S. (2006) Desenvolvimento de uma aplicação colaborativa móvel e sensível ao contexto para a plataforma Infraware. Monografia de graduação, Universidade Federal do Espírito Santo, Vitória, Brasil.

MORIKAWA, D., HONJO, M., MASAYOSHI, A,Y. (2004) Profile Aggregation and Dissemination : A Framework for Personalized Service Provisioning.

MOTAKIS, I., ZANIOLO, C. (1995) A Formal Semantics for Composite Temporal Events in Active Database Rules. Journal of Systems Integration

MUÑOZ, M. A, RODRÍGUEZ, M., FAVELA, J., MARTINEZ-GARCIA, A. I., GONZÁLEZ, V., M. (2003) Context-aware mobile communication in hospitals. IEEE Computer, Vol 36, No. 9, 38-46.

NADAI, F. P. (2006) Acesso e integração de dados para sistemas context-aware. Projeto de graduação. Departamento de Informática, UFES, Brasil, 2006.

NATH, S. (2002) Historical Queries on IrisNet. Disponível em < www.intel-iris.net/papers/pervasive-03.pdf>

OLIVEIRA, N.Q., BARBOSA, A.C.P. (2007) Uma Arquitetura para Integração de Dados em Sistemas Sensíveis ao Contexto. Workshop on Tools and Applications – Webmedia 2007, Gramado, RS, Brasil

ORACLE (2006) Disponível em http://www.oracle.com

OZSOYOGLU, G., SNODGRASS, R.T. (1995) Temporal and Real-time databases: a survey. Disponível em < ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=404027 >

ÖZSU, M.T.; VALDURIEZ, P. (2001) Princípios de Sistemas de Banco de Dados Distribuídos. 2a edição, Editora Campus, Rio de Janeiro – RJ.

PASCOE, J. (2001) Context-aware software. Tese de doutorado, University of Kent, 2001. Disponível em < http://www.cs.kent.ac.uk/pubs/2001/1390/index.html >

PATON, N., DIAZ, O. (1999) Active Database Systems. ACM Computing Surveys, 1999

PAYTON et. al. (2005) A query-centered perspective on context awareness in mobile Ad Hoc Networks.

PEREIRA FILHO, J.G.P *et. al*. (2006) Infraware: um middleware de suporte à aplicações sensíveis ao contexto. Artigo resumido. Simpósio Brasileiro de Redes de Computadores, Curitiba, Brasil, 2006

PEREIRA FILHO, J. G., BARBOSA, A. C. P. , CALVI, C. Z., MANTOVANELLI, R. P., MONTEIRO, R. R., OLIVEIRA, N. Q. (2006) Arquitetura da plataforma INFRAWARE. Relatório Técnico.

PEREIRA FILHO, J. G. ; BARBOSA, A. C. P. ; CALVI, C. Z. ; MANTOVANELLI, R. P. ; MONTEIRO, R. R. ; OLIVEIRA, N. Q. (2005). Plataformas de Serviços Context-aware. Relatório Técnico

PERICH, F. et al. (2004) On Data Management in Pervasive Computing Environments. University of Maryland, Baltimore County, 2004.

PERICH, F., et.al. (2005) Collaborative Joins in a Pervasive Computing Environment. Department of Computer Science & Electrical Engineering University of Maryland, 2005.

PERNAS, A. M. (2004) Ontologias aplicadas à descrição de recursos em grids computacionais. Dissertação de mestrado, Universidade Federal de Santa Catarina, 2004.

PESSOA, R.M. (2006) Infraware: Um Middleware de Suporte a Aplicações Sensíveis ao Contexto. 2006. Dissertação de mestrado, Universidade Federal do Espírito Santo, 2006.

PESSOA, R. M., CALVI, C.Z., PEREIRA FILHO, J.G., ANDRIAO, R.V. (2006) Aplicação de um Middleware Sensível ao Contexto em um Sistema de Telemonitoramento de Pacientes Cardíacos. In: SEMISH - Seminário Integrado de Software e Hardware, 2006, Campo Grande - MS. Anais do XXXIII SEMISH - XXVI Congresso da Sociedade Brasileira de Computação,

PIETZUCH, P.R. (2004) Hermes: A Scalable event-based middleware. Relatório técnico, 2004. Disponível em < www.cl.cam.ac.uk/techreports/UCAM-CL-TR-590.pdf >

PIETZUCH, P. R., SHAND, B., BACON, J. (2003) A framework for event composition in distributed systems. Proceedings of the 4th International Conference on Middleware, Rio, Brasil, 2003

PINHEIRO, F.S. (2004) Incorporando uma máquina de execução de consultas ao CoDIMS. Monografia de Graduação, Universidade Federal do Espírito Santo, Vitória, 2004.

PITOURA, E., STEFANIDIS, K., (2004) Context in databases. Relatório técnico. Department of Computer Science, University of Ioannina

POKRAEV, S. *et. al.* (2003) Context-aware services: state of the art

POSTGRESQL (2006) Disponível em <http://www.postgresql.org >

PRESSMAN, R. (2001) Engenharia de Software. ISBN: 85-86804-57-6. Ed. McGraw-Hill

RAKOTONIRAINY, A., INDULSKA, J, LOKE, S.W., ZASLAVSKY, A. (2001) Middleware for reactive components: An integrated use of context, roles, and event based coordination

REY, G., COUTAZ, J. (2004) Contextor: capture and dynamic distribution of contextual information. ACM International Conference Proceeding Series; Vol. 64 archive. Proceedings of the 1st French-speaking conference on Mobility and ubiquity computing

ROBIN, A. (2006) Using SensorML to describe a Complete Weather Station. Tutorial

ROUSSOS, Y., ZOUMBOULAKIS, M. (2004) Ubiquitous computing and databases: critical issues and challenges. L.Rovero (ed.) Encyclopaedia of Databases, Idea Group Inc, Hershey, PA, USA.

ROUSSOS, Y., STAVRAKAS, Y., PAVLAKI, V. (2005) Towards a context-aware relational model. Disponível em < sra.itc.it/events/crr05/190.pdf >

SACRAMENTO, V. et. al. (2004) An Architecture Supporting the Development of Collaborative Applications for Mobile Users. Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE-2004)

SAX (2007). SAX API. Disponível em <http://www.saxproject.org>

SBC (2006) Grandes Desafios da Pesquisa em Computação no Brasil.
Disponível em < www.sbc.org.br/index.php?language=1&content=downloads&id=272 >

SCHILIT, B., ADAMS, N., WANT, R. (1994) Context-aware computing applications. IEEE Workshop on Mobile Computing Systems and Applications (WMCSA´94), Santa Cruz, 89-101.

SCHWIDERSKI, S. (1996) Monitoring the behavior of distributed systems. Tese de doutorado, University of Cambridge, 1996.

SENSORML (2006) Disponível em: <http://vast.nsstc.uah.edu/SensorML/>

SHETH, A.,P., LARSON, J. A. (1990) A. Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput. Surv., ACM Press, v. 22, n. 3, p. 183-236, 1990. ISSN 0360-0300.

SILVESTRE, L.J. (2005) Uma Abordagem Baseada em Ontologias para a Gerencia de Metadados do CoDIMS. Dissertação de mestrado, Universidade Federal do Espírito Santo, 2005.

SINDEREN, M.J. et. al. (2006) Overall architecture of the AWARENESS infrastructure. Relatório tecnico. Disponível em
<http://www.freeband.nl/FreebandKC/documents?keyword_id=2432>

STEFANDIS, K., PITOURA, E.,VASSILIADIS, P. (2005) On Supporting Context-Aware Preferences in Relational Database Systems. Disponível em
< www.cs.uoi.gr/~pvassil/publications/2005_MCMP/MCMP_2005.pdf >

TELECARDIO (2005): Projeto Telecardio - Telecardiologia a Serviço do Paciente em Ambientes Hospitalares e Residenciais. DI/DEE/UFES, Financiamento: FAPES.

TOENNIS, M. (2003) Data Management for Augmented Reality Applications. Dissertação de mestrado. Technische Universitaet Muenchen.

TOMASIC, A., SIMON, E. (1997) Improving access to Environmental data using context information. SIGMOD Record Volume 26, Number 1, 1997

TREVISOL, G.G. (2004) CoDIMS: Incorporando uma nova abordagem na comunicação entre seus componentes. Monografia de Graduação, Universidade Federal do Espírito Santo, Vitória, 2004.

TREVISOL, G.G., BARBOSA, A.C.P. (2006) Uma proposta de execução distribuída de consultas em um ambiente de grid para o CoDIMS. IV Workshop on Computational Grids and Applications (WCGA 2006), Curitiba, 2006.

TREVISOL, G.G. et. al. (2007). A Distributed Query Execution Engine in a Grid Environment. CCGrid 2007: Seventh IEEE International Symposium on Cluster Computing and the Grid 2007. Rio de Janeiro, Brasil.

UML (2007) Unified Modeling Language. Disponível em < http://www.uml.org>

VARGAS, L., BACON, J., MOODY, K. (2005) Integrating databases with publish/subscribe. Proceedings of the 25th IEEE ICDCSW' 05, 2005.

VASTENBURG, M. (2004) Situated profiles for aware environments. Proceedings in First interntional Workshop on Computer support for human tasks and activities

VOLZ, S., SESTER, M. (2000) NeXus – Distributed Data Management concepts for Location-aware Applications. Disponível em http://www.ifp.uni-stuttgart.de/publications/2000/Volz_Ascona_00.pdf

W3C, Web Service; Disponível em: <http://www.w3.org/2002/ws/ >

W3C, XML; World Wide Web Consortium. Disponível em: http://www.w3.org/XML

W3SCHOOLS (2006) Disponível em <http://www.w3schools.com/soap/default.asp>

WASP Project (2003) Disponível em <http://www.freeband.nl/projecten/wasp/ENindex.html>

WEGDAM, M. (2005) AWARENESS: a project on Context AWARE NEtworks and ServiceS, Proceedings of the 14th Mobile & Wireless Communications Summit 2005, 19-23 June 2005, Dresden, Germany

WEISER, M. (1991) The Computer for the Twenty-First Century. Scientific American, pp. 94-10, September, 1991

WIDOW, J. (1996) The Starburst active database rule system. IEEE Transactions on Knowledge and Data Engineering, volume 8, issue 4, pages 583-595

YU, S., AL-JADIR, L., SPACCAPIETRA, S. (2005) Matching User's Semantics with Data Semantics in Location-Based Services. 1st Workshop on Semantics in mobile Environments (SME 2005), Ayia Napa, 2005. Disponível em <ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-165/paper16.pdf>

ZIEGLER, P., DITTRICH, K.R. (2004): Three Decades of Data Integration - All Problems Solved. IFIP Congress Topical Sessions 2004, 3-12. Toulouse, France. Disponível em arvo.ifi.unizh.ch/dbtg/Staff/Ziegler/papers/ZieglerWCC2004.pdf

Anexo A
Código fonte do operador JOIN

```
package ProcConsulta2;
import java.util.*;
import DAO.DAO;
import org.apache.soap. *;
import java.net.*;

public class JoinRelacional implements OperadorComposto,  java.io.Serializable {

private String tabelaEsq = "";
private String tabelaDir = "";
private Resultset entradaEsq = new Resultset();
private Resultset entradaDir= new Resultset();

private int ordemExecucao;
private Vector produtores = new Vector();
private int consumidor;
private Vector condicoes = new Vector();
private Vector atributos = new Vector();
private Resultset saida = new Resultset();
private int proximoOperador;
private int container = 0;
private int containerProximo = 0;
private  boolean raiz = false;

public JoinRelacional2() {}

public void executar()
{
    Vector tuplas = new Vector();
    for(int e = 0; e < entradaEsq.getTuplas().size();e++)
        for(int d = 0; d < entradaDir.getTuplas().size(); d++)
            {
            Tupla te = (Tupla) entradaEsq.getTuplas().get(e);
            Tupla td = (Tupla) entradaDir.getTuplas().get(d);
            if(te.getValores().get(0).equals(td.getValores().get(0)))
{

                // Faz o merge de todos os atributos das duas entradas
                Tupla t = new Tupla();
                Vector nomes = new Vector();
                Vector valores = new Vector();
                for(int g = 0; g < te.getNomes().size(); g++) {
                    nomes.add(te.getNomes().get(g));
                    valores.add(te.getValores().get(g));  }
                for(int h = 1; h < td.getNomes().size();h++)  {
                    nomes.add(td.getNomes().get(h));
                    valores.add(td.getValores().get(h));  }

                t.setNomes(nomes);t.setValores(valores);
                tuplas.add(t);
                }
            }
    saida.setTuplas(tuplas);
    System.out.println(this.saidaToString());

    DAO dao = new DAO();
    dao.inserirResultset(saida);
```

```
    // Retorna o controle para a MEC para executar outra subscricao
    if(this.proximoOperador == 999)
         try {
         Call call = new Call();
         Vector params2 = new Vector();
         call.setTargetObjectURI("urn:MEC");
         call.setMethodName("retornarControleMEC");
         call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
         call.setParams(params2);

  Response
resp2 = call.invoke(new URL("http://localhost:8090/soap/servlet/rpcrouter"), "");
    catch(Exception e) {e.printStackTrace();}
    }   }

 public void setarEntradaEsq(String s)
    {
    DAO dao = new DAO();
    System.out.println("HASH CODE:"+s);
    Resultset aux = dao.consultarResultset(Integer.parseInt(s));
    entradaEsq = aux;
    }

 public void setarEntradaDir(String s)
    {
    DAO dao = new DAO();
    Resultset aux = dao.consultarResultset(Integer.parseInt(s)); entradaDir = aux}
```

Anexo B
Código fonte de um agente *event-driven*

```
package Agente;

import org.jdom.*;
import org.jdom.input.SAXBuilder;

import ProcConsulta2.Resultset;
import ProcConsulta2.Tupla;

import ProcConsulta2.Operador2;
import ProcConsulta2.OperadorSimples;

import java.io.File;
import java.net.URL;
import java.util.*;
import org.apache.soap.Constants;
import org.apache.soap.rpc.*;

import org.apache.soap.encoding.*;
import org.apache.soap.encoding.soapenc.*;
import org.apache.soap.util.xml.*;

import ProcConsulta2.*;

public class AgenteED extends Thread implements InterfaceAgente,
java.io.Serializable {

private int lifetime = 0;
private EventoSimples evento;
private int elapsedTime = 0;
private String subConsulta = "";

Resultset rs = new Resultset();
ArrayList historicos = new ArrayList();
Resultset ids = new Resultset();

 public int operadorPai;

 private int container = 0;
 private int containerOperadorPai = 0;

 private String nomeFonte = "";

//public void iniciarMonitoramento() {this.run();}

public void run() {

try {
    elapsedTime = 0;
    // enquanto o status do evento for "espera" E o timeout da subscrição não
ocorrer
    while(evento.getStatus() == 1 && elapsedTime < lifetime)
          {
 //    OperadorSimples os = (OperadorSimples) operadorPai;
```

```
 //  String nomeFonte =  os.getTabela().substring(0, os.getTabela().length() - 5);

    Vector params = new Vector();
    params.add(new Parameter("astr_pathBaseDesembarque", String.class,
"/usr/codims/CoDIMS/FonteDados/" + getNomeFonte() + "/", null));
    Response resp = chamarWebService("urn:wrapper"+ getNomeFonte().toLowerCase(),
"setPathBaseDesembarque", params);
    System.out.println("/usr/codims/CoDIMS/FonteDados/" + getNomeFonte());


    Vector params1 = new Vector();
    String subConsulta = "<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?><local-
plano>" +"<coluna ordem=\"0\" nome=\" cod_paciente\" tabela=\"" + getNomeFonte()
+"\" />" + "<coluna ordem=\"1\" nome=\" " + evento.getArgumentoEsquerdo() + "\"
tabela=\"" + getNomeFonte() + "\" />" +"<tabela nome=\"" + getNomeFonte() + "\"/>"
+ "</local-plano>";
    params1.add(new Parameter("subConsulta", String.class, subConsulta, null));
    Response resp1 = chamarWebService("urn:wrapper"+ getNomeFonte().toLowerCase(),
"executarSubConsulta", params1);


    // le o resultset
    Resultset rs = new Resultset();

    File f = new File(resp1.getReturnValue().getValue()+"");
    SAXBuilder sb = new SAXBuilder();
    System.out.println(f.getPath() + f.getName());
    Document d = sb.build(f);
    Element raiz = d.getRootElement();

    List elementos = raiz.getChildren();
    for(int i = 0; i < elementos.size(); i++)
        {Element e = (Element) elementos.get(i);
         List atr = e.getAttributes();
         Tupla t = new Tupla();
         for(int j = 0; j < e.getAttributes().size();j++)
            {
            Attribute a = (Attribute) atr.get(j);
            t.getNomes().add(a.getName());
            t.getValores().add(a.getValue());
             }
         rs.getTuplas().add(t);
        }
        System.out.println("rs " + rs.getTuplas().size() + " tuplas");

    // Seleciona a quantia de historico
    historicos.add(new Resultset());
    historicos.add(new Resultset());

    Resultset historico = (Resultset) historicos.get(0);

    // Obtem qual o indice do atributo a ser comparado no resultset

    Tupla t = (Tupla) rs.getTuplas().get(0);
    int indice = t.getNomes().indexOf(evento.getArgumentoEsquerdo());

    // faz as comparações
     for(int k = 0; k < historico.getTuplas().size();k++)
        {

        Tupla t1 = (Tupla) rs.getTuplas().get(k);
        Tupla t2 = (Tupla) historico.getTuplas().get(k);
        String i1 = t1.getValores().get(indice)+"";
        String i2 = t2.getValores().get(indice)+"";
```

```
        if(i1.compareTo(i2) > 0)
            {


            // Adiciona à lista para
            Tupla novaTupla = new Tupla();
            novaTupla.getValores().add(t1.getValores().get(0));
            novaTupla.getValores().add(t1.getValores().get(indice));
            novaTupla.getNomes().add(t1.getNomes().get(0));
            novaTupla.getNomes().add(t1.getNomes().get(indice));
            ids.getTuplas().add(novaTupla);

            evento.setStatus(2);

            historicos.set(1,historicos.get(0));
            historicos.set(0,rs);

            try {

         Call call = new Call();
         Vector params2 = new Vector();
        call.setTargetObjectURI("urn:container" + containerOperadorPai);
        call.setMethodName("recebeMensagem");
        call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);

         SOAPMappingRegistry smr = new SOAPMappingRegistry();
       BeanSerializer bsr = new BeanSerializer();
       QName qn = new QName("urn:procConsulta2", "ProcConsulta2.Resultset");
       smr.mapTypes(Constants.NS_URI_SOAP_ENC, qn, Resultset.class, bsr, bsr);

       BeanSerializer bsr3 = new BeanSerializer();
       QName qn3 = new QName("urn:procConsulta2", "ProcConsulta2.Tupla");
       smr.mapTypes(Constants.NS_URI_SOAP_ENC, qn3, Tupla.class, bsr3, bsr3);

       call.setSOAPMappingRegistry(smr);

        params2.add(new Parameter("id", Integer.class, operadorPai, null));
      params2.add(new Parameter("saida", Resultset.class, ids, null));

        call.setParams(params2);
        Response resp2 = call.invoke(new
URL("http://localhost:8090/soap/servlet/rpcrouter"), "");
 ())
}
catch(Exception e) {e.printStackTrace(); }

            }
        }
       historicos.set(1,historicos.get(0));
       historicos.set(0,rs);

       elapsedTime = elapsedTime + 15000;
        sleep(15000);

    } // while

    // encerra a thread

}
catch(Exception e){
    e.printStackTrace();}
}

public void pausarMonitoramento()
{
```

```
try {
    this.wait();
    }
catch(Exception e)
    {e.printStackTrace();}
}

public void interromperMonitoramento()
{
try {
    this.interrupt();
   // this.finalize();
    }
catch(Exception e)
    {e.printStackTrace();}

}

}

public void setSubConsulta(String subConsulta) {
          this.subConsulta = subConsulta;
}

public String toString()
{
 //   return "agente " + this.evento.toString();
    String s = "AGENTE ";
    for(int y = 0; y < ids.getTuplas().size();y++) {
        Tupla t = (Tupla) ids.getTuplas().get(y);
        s = s + " " + t.getValores().get(0); }
     return s;
}


public Response chamarWebService(String uri, String nomeMetodo,  Vector params)
{
try {
    Call call = new Call();
    call.setTargetObjectURI(uri);
    call.setMethodName(nomeMetodo);
    call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
    call.setParams(params);
    Response resp = call.invoke(new
URL("http://localhost:8090/soap/servlet/rpcrouter"), "");
    Parameter result = resp.getReturnValue();
    return resp;
    }
catch(Exception e) {e.printStackTrace();}
   return null;
}

. . .
Setters e getters
}
```

Anexo C
Arquivo de configuração

```
<configuracao><hosts>
<nome_comp=”AcessoDados” host=”127.0.0.1” porta=”9090” uri=”urn:acessodados”>
<nome_comp=”MetaDados” host=”127.0.0.1” porta=”9090” uri=”urn:metadados”>
<nome_comp=”ProcConsulta” host=”127.0.0.1” porta=”9090” uri=”urn:procconsulta”>
<nome_comp=”Rmec” host=”127.0.0.1” porta=”9090” uri=”urn:rmec”>

<nome_comp=”FabricaAgentes” host=”127.0.0.1” porta=”9090” uri=”urn:fabricaAgentes”>
<nome_comp=”Historico” host=”127.0.0.1” porta=”9090” uri=”urn:historico”>
<nome_comp=”Perfil” host=”127.0.0.1” porta=”9090” uri=”urn:perfil”>
<nome_comp=”EventReport” host=”127.0.0.1” porta=”9090” uri=”urn:eventReport”>

<nome_comp=”GerenciaFontes” host=”127.0.0.1” porta=”9090” uri=”urn:gerenciaFontes”>
<nome_comp=”GerenciaTaref” host=”127.0.0.1” porta=”9090” uri=”urn:gerenciaTarefas”>
<nome_comp=”Container” host=”127.0.0.1” porta=”9090” uri=”urn:container”>

<nome_comp=”WrapperPacien” host=”127.0.0.1” porta=”9090” uri=”urn:wrapperPaciente”>
<nome_comp=”WrapperTemperatura” host=”127.0.0.1” porta=”9090”
uri=”urn:wrapperTemperatura”>
<nome_comp=”WrapperECG” host=”127.0.0.1” porta=”9090” uri=”urn:wrapperECG”>

    <configuracao_fisica>
        <operacoes_oferecidas>
            <componente_ofer nome_comp="AcessoAosDados">
                <operacao_ofer nome_ope="executar_sub_consulta" />
                <operacao_ofer nome_ope="obter_proximo_dado" />
            </componente_ofer>

            <componente_ofer nome_comp="Metadados">
                <operacao_ofer nome_ope="definir_metadados" />
                <operacao_ofer nome_ope="exibir_metadados" />
                <operacao_ofer nome_ope="obter_objeto_MD" />
            </componente_ofer>

            <componente_ofer nome_comp="ProcessamentoDeConsulta">
                <operacao_ofer nome_ope="analisar_consulta" />
                <operacao_ofer nome_ope="reescrever_consulta" />
                <operacao_ofer nome_ope="otimizar_consulta" />
                <operacao_ofer nome_ope="processar_consulta" />
                <operacao_ofer nome_ope="criarPlanosLocais"/>
            </componente_ofer>

            <componente_ofer nome_comp="Perfil">
                <operacao_ofer nome_ope="personalizarConsulta"/>
                <operacao_ofer nome_ope="consultarPerfil"/>
            </componente_ofer>

            <componente_ofer nome_comp="Historico">
                <operacao_ofer nome_ope="armazenarHistorico"/>
                <operacao_ofer nome_ope="obterDadosHistoricos"/>
                <operacao_ofer nome_ope="iniciarArmazenamento”/>
                <operacao_ofer nome_ope="pausarArmazenamento”/>
            </componente_ofer>

             <componente_ofer nome_comp="Gerenciafontes”>
                <operacao_ofer nome_ope="checarTodasFontes”>
                <operacao_ofer nome_ope="checarFonte”>
```

```
                <operacao_ofer nome_ope="monitorarStatusFontes”>
                <operacao_ofer nome_ope="alertarDesconexao”>
            </componente_ofer>

            <componente_ofer nome_comp="Gerenciatarefas”>
                <operacao_ofer nome_ope="checarMaiorPrioridade”>
                <operacao_ofer nome_ope="adicionarSubscricao”>
                <operacao_ofer nome_ope="removerSubscricao”>
            </componente_ofer>

            <componente_ofer nome_comp="Agente”>
                <operacao_ofer nome_ope="iniciarMonitoramento”>
                <operacao_ofer nome_ope="pausarMonitoramento”>
                <operacao_ofer nome_ope="pararMonitoramento”>
                <operacao_ofer nome_ope="run”>
            </componente_ofer>

            <componente_ofer nome_comp="Subscrição”>
                <operacao_ofer nome_ope="popularSubscricao”>
            </componente_ofer>

        <operacoes_requeridas>
            <componente_requer nome_comp="ProcessamentoDeConsulta">
               <operacao_requer nome_comp="Metadados" nome_ope="obter_objeto_MD" />
            </componente_requer>

            <componente_requer nome_comp="ProcessamentoDeConsulta">
                <operacao_requer nome_comp="AcessoAosDados"
nome_ope="obter_proximo_dado" />
            </componente_requer>
                </configuracao_fisica>
</configuracao>


</hosts><configuracao>
```

Anexo D
Dicionário de dados

Contexto: qualquer informação que pode ser usada para caracterizar uma situação de uma entidade. Uma entidade é uma pessoa, um lugar, ou um objeto que é considerado relevante para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a própria aplicação [DEY, 2001].

Evento: ação instantânea e atômica, ou seja, ela ocorre um ponto especifico no tempo. Representa uma ação ocorrida no mundo real, com semântica e conseqüências. Um evento pode se encontrar em vários estados, tais como criado, em espera, gerador e disparado.

***Framework*:** é uma arquitetura semi-completa e reutilizável que pode ser instanciada para produzir aplicações customizadas [FAYAD, 1999].

Máquina de execução de consulta: componente de um sistema gerenciador de banco de dados responsável pela execução seqüencial das instruções encontradas em um plano de execução de consultas.

Plano de execução de consulta: conjunto de operações para a realização de uma consulta, normalmente baseadas na álgebra relacional e passível de otimização.

Subscrição: consultas realizadas à plataforma DBMWare, traduzida em um plano de execução de consulta.

Anexo E
DTD para a validação da PEC orientada a eventos

```
<!ELEMENT plano (operador+)>
<!ELEMENT operador ((nao | e | ou
| aoAumentar | aoDiminuir | maior | menor | igual | soma | contador), produtor*, consumidor*)>

<!ELEMENT operador ((antesDe | depoisDe | disjuntos
| aoRepetir | comDuração ), produtor*, consumidor*)>

<!ELEMENT operador ((desde    | antesDe (data) | entre | após (frequencia) |    horários (agenda) | ), produtor*,
consumidor*)>

<!ELEMENT operador ((aoObterDados    |aoAtualizarDados | aoRemoverDados ), produtor*, consumidor*)>

<!ATTLIST operador ordem-execucao CDATA #REQUIRED>
<!ELEMENT produtor EMPTY>
<!ATTLIST produtor op-produtor CDATA #REQUIRED>
<!ELEMENT consumidor EMPTY>
<!ATTLIST consumidor
op-consumidor CDATA #REQUIRED>


<!ELEMENT scan-relacional (condicao*, coluna*)>
<!ATTLIST scan-relacional
tabela CDATA #REQUIRED>
<!ELEMENT project-relacional (coluna+)>
<!ELEMENT join-nested-loop-relacional (condicao*)>
<!ATTLIST join-nested-loop-relacional
tabela-esquerda CDATA #REQUIRED
tabela-direita CDATA #REQUIRED>
<!ELEMENT coluna EMPTY>
<!ATTLIST coluna
nome CDATA #REQUIRED
tabela-origem CDATA #REQUIRED>
<!ELEMENT condicao EMPTY>
<!ATTLIST condicao
argumento-esquerdo CDATA #REQUIRED
operacao (nao | igual | diferente | maior | menor | maior-igual | menorigual)
#REQUIRED
argumento-direito CDATA #REQUIRED>
<!ELEMENT scan-xml ((elemento+), (condicaoXML*))>
<!ATTLIST scan-xml arquivo CDATA #REQUIRED>
<!ELEMENT project-xml (elemento+)>
<!ELEMENT join-nested-loop-xml (condicaoXML*)>
<!ATTLIST join-nested-loop-xml arquivo-esquerda CDATA #REQUIRED arquivo-direita CDATA #REQUIRED>
<!ELEMENT elemento (atributo*)>
<!ATTLIST elemento caminho CDATA #REQUIRED obter-conteudo (sim | nao) #REQUIRED>
<!ELEMENT atributo EMPTY>
<!ATTLIST atributo
nome CDATA #REQUIRED>
```

Anexo F
DTD de subscrições

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.w3schools.com"
  xmlns="http://www.w3schools.com"
  elementFormDefault="qualified">

<xs:element name="subscription">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="substype">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="Request-Response"/>
      <xs:enumeration value="Time-Driven"/>
      <xs:enumeration value="Event-Driven"/>
      <xs:enumeration value="Continuously"/>
     </xs:restriction>
    </xs:simpleType>

   </xs:element>
   <xs:element name="appid" type="xs:integer"/>
   <xs:element name="userid" type="xs:integer"/>
   <xs:element name="usergroup" type="xs:string"/>
   <xs:element name="lifetime" type="xs:integer"/>
   <xs:element name="sendrate" type="xs:integer" minOccurs="0"/>
   <xs:element name="eventslist" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="event" maxOccurs="unbounded">
       <xs:complexType>
        <xs:sequence>
        <xs:element name="scan-Relacional">
         <xs:complexType>
          <xs:sequence>
           <xs:attribute name="tabela" type="xs:string"/>
            <xs:element name="coluna" minOccurs="1" maxOccurs="unbounded">
             <xs:complexType>
              <xs:sequence>
               <xs:attribute name="nome" type="xs:string"/>
               <xs:attribute name="tabela-origem" type="xs:string"/>
              </xs:sequence>
             </xs:complexType>
            </xs:element>
            <xs:element name="ladoEsquerdo" type="xs:string"/>
            <xs:element name="operador"    type="xs:string"/>
            <xs:element name="ladoDireito"  type="xs:string"/>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
```

```
      </xs:complexType>
     </xs:element>
    <xs:element name="conector" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:attribute name="tipo" type="xs:string"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="actionslist" minOccurs="1">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="action" maxOccurs="unbounded">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="scan-Relacional">
         <xs:complexType>
          <xs:sequence>
           <xs:attribute name="tabela" type="xs:string"/>
           <xs:element name="coluna" minOccurs="1" maxOccurs="unbounded">
            <xs:complexType>
             <xs:sequence>
              <xs:attribute name="nome" type="xs:string"/>
              <xs:attribute name="tabela-origem" type="xs:string"/>
             </xs:sequence>
            </xs:complexType>
           </xs:element>
           <xs:element name="ladoEsquerdo" type="xs:string"/>
           <xs:element name="operador"    type="xs:string"/>
           <xs:element name="ladoDireito"  type="xs:string"/>
           <xs:element name="acao"      type="xs:tring" minOccurs="0"
maxOccurs="unbounded"/>
          </xs:sequence>
         </xs:complexType>
        </xs:element>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="serviceslist" minOccurs="0">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="service" type="xs:string" maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
</xs:element>

</xs:schema>
```

.

Anexo G

Representação OWL dos metadados

```
<owl:FunctionalProperty rdf:ID="fontName"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="type"><owl:oneOf rdf:parseType="Collection">
<type rdf:about="#Relational"/>
<type rdf:about="#XML"/>
<type rdf:about="#GPS"/><owl:oneOf/></owl:FunctionalProperty>
<owl:DatatypeProperty rdf:ID="url"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="port"></owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="driver"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="availability></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="schedule”></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="refreshRate"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="isAutoSend"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="maximumGap"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="validity”></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="hasReplica"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="acuracy"></owl:FunctionalProperty>

<rdfs:comment>Aspects related to context-awareness</rdfs:comment>
<owl:FunctionalProperty rdf:ID="contextType"><owl:oneOf>
<contextType rdf:about="#Static"/>
<contextType rdf:about="#Sensed"/>
<contextType rdf:about="#Derived"/>
<contextType rdf:about="#Profile"/><owl:oneOf/> </owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="storage"><owl:oneOf rdf:parseType="Collection">
<contextType rdf:about="#physical"/>
<contextType rdf:about="#virtual"/>
<contextType rdf:about="#hybrid"/><owl:oneOf/></owl:FunctionalProperty>

<rdfs:comment>Statistics usefull for query optimization</rdfs:comment>
<owl:FunctionalProperty rdf:ID="accessCount"></owl:FunctionalProperty>
<rdfs:comment>Provenance data</rdfs:comment>
<owl:FunctionalProperty rdf:ID="author"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="owner"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="goal"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="version"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="startDate"></owl:FunctionalProperty>

<rdfs:comment>Aspects related to device capacity</rdfs:comment>
<owl:FunctionalProperty rdf:ID="OS"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="RAM"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="CPU"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="battery"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="location"></owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="network"><owl:oneOf rdf:parseType="Collection">
<contextType rdf:about="#bluetooth"/>
<contextType rdf:about="#WI-FI"/>
<contextType rdf:about="#TCP/IP"/></owl:FunctionalProperty>
```